Classes, Objects, and CSV Data Processing

Lecture 12

```
3  class Genotype {
4      alleles: string = "";
5  }
6
7  let main = async () => {
8      let mom = new Genotype();
9      mom.alleles = "Bb";
10
11     let dad = new Genotype();
12     dad.alleles = "bb";
13
14     let child = cross(dad, mom);
15     print(child.alleles);
16 };
17
18 let cross = (a: Genotype, b: Genotype): Genotype => {
19     let child = new Genotype();
20     let aAllele = a.alleles[0];
21     let bAllele = b.alleles[0];
22     child.alleles = aAllele + bAllele;
23     return child;
24 };
25
26 main();
```

Challenge Question 1. Draw an environment diagram of the code listing and respond to the sequence of questions on PollEverywhere once completed.

Hands-on: Simulating Genotype Inheritance

- In this program you'll repeat a simple experiment that crosses two parent genotypes with one another and count the number of times the dominant gene is expressed in offspring.

- Open lec11/00-genotype-app.ts - at the TODO:

1. Write a loop that iterates **trials** (a variable) number of times
2. Inside the loop, declare a variable named **child** that is assigned the value returned by crossing **mom** and **dad** with the **cross** function.
3. Call the **hasDominantAllele** function with the **child** and **"B"** as arguments ("B" being the dominant allele). *If* it returns true, then increase the **dominants** count variable by 1.
4. After the loop, compute the ratio of trials which resulted in offspring with a dominant B allele. Print this value.

- Done - Check-in on PollEv.com/compunc
  - Then: try increasing # of trials
  - Try changing one of the parent alleles to be BB or bb and comparing outcomes.

```
// TODO: Write a loop that repeats trials # of times
for (let i = 0; i < trials; i++) {
    let child = cross(mom, dad);
    if (hasDominantAllele(child, "B")) {
        dominants += 1;
    }
}
print("Brown Eyes: " + (dominants / trials));
```

```
3  class Point {
4      x: number = 0;
5      y: number = 0;
6  }
7
8  export let main = async () => {
9      let a: Point = new Point();
10     a.x = 2;
11     a.y = 3;
12     let b = transX(a, 1);
13     let c = scale(a, 2);
14     print(a.x);
15     print(b.x);
16     print(c.x);
17 };
18
19 let scale = (p: Point, by: number): Point => {
20     p.x *= by;
21     p.y *= by;
22     return p;
23 };
24
25 let transX = (p: Point, dx: number): Point => {
26     let t = new Point();
27     t.x = p.x + dx;
28     t.y = p.y;
29     return t;
30 };
31
32 main();
```

Challenge Question . Draw an environment diagram of the code listing and respond to the sequence of questions on PollEverywhere once completed.

Avoid Writing Functions that Mutate Parameters

```
let scale = (p: Point, by: number): Point => {
    p.x *= by;
    p.y *= by;
    return p;
};

let transX = (p: Point, dx: number): Point => {
    let t = new Point();
    t.x = p.x + dx;
    t.y = p.y;
    return t;
};
```

- The differences between the **scale** and **transX** functions in the previous example are subtle but very important to understand.
- The **scale** function *mutates* or "changes" the referenced object it is given.
- The **transX** function does not modify the referenced object it is given and instead returns a reference to a new object.
- Generally avoid writing functions like **scale** which mutate objects or arrays passed in by reference.
  - Why? It's much harder to reason about and debug functions that mutate reference parameters.
  - When is it ok or necessary to mutate reference params? When the cost of creating copies is very high.

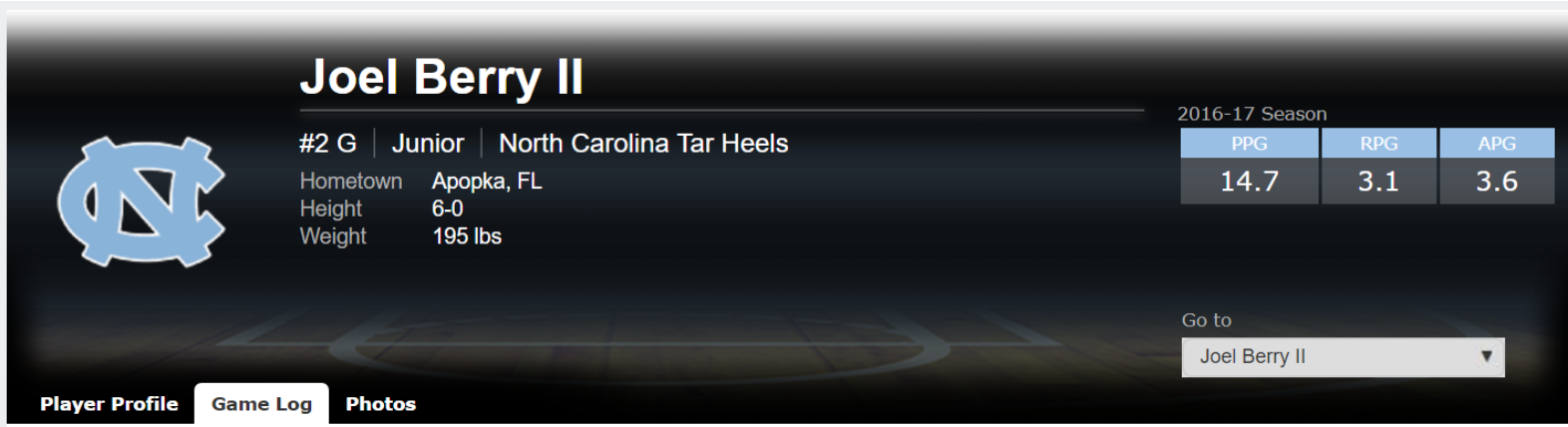
```
class Point {
    x: number = 0;
    y: number = 0;
}

export let main = async () => {
    let a: Point = new Point();
    a.x = 2;
    a.y = 3;
    let b = transX(a, 1);
    let c = scale(a, 2);
    print(a.x);
    print(b.x);
    print(c.x);
};

let scale = (p: Point, by: number): Point => {
    p.x *= by;
    p.y *= by;
    return p;
};

let transX = (p: Point, dx: number): Point => {
    let t = new Point();
    t.x = p.x + dx;
    t.y = p.y;
    return t;
};

main();
```

Working with Data

- **Let's work with Joel Berry II's game data from UNC's 2016-17 championship season.**

Today's Data

- Data source: ESPN.com
- The Game Log table to the right was copied and pasted into Excel



Joel Berry II Game-by-Game Stats

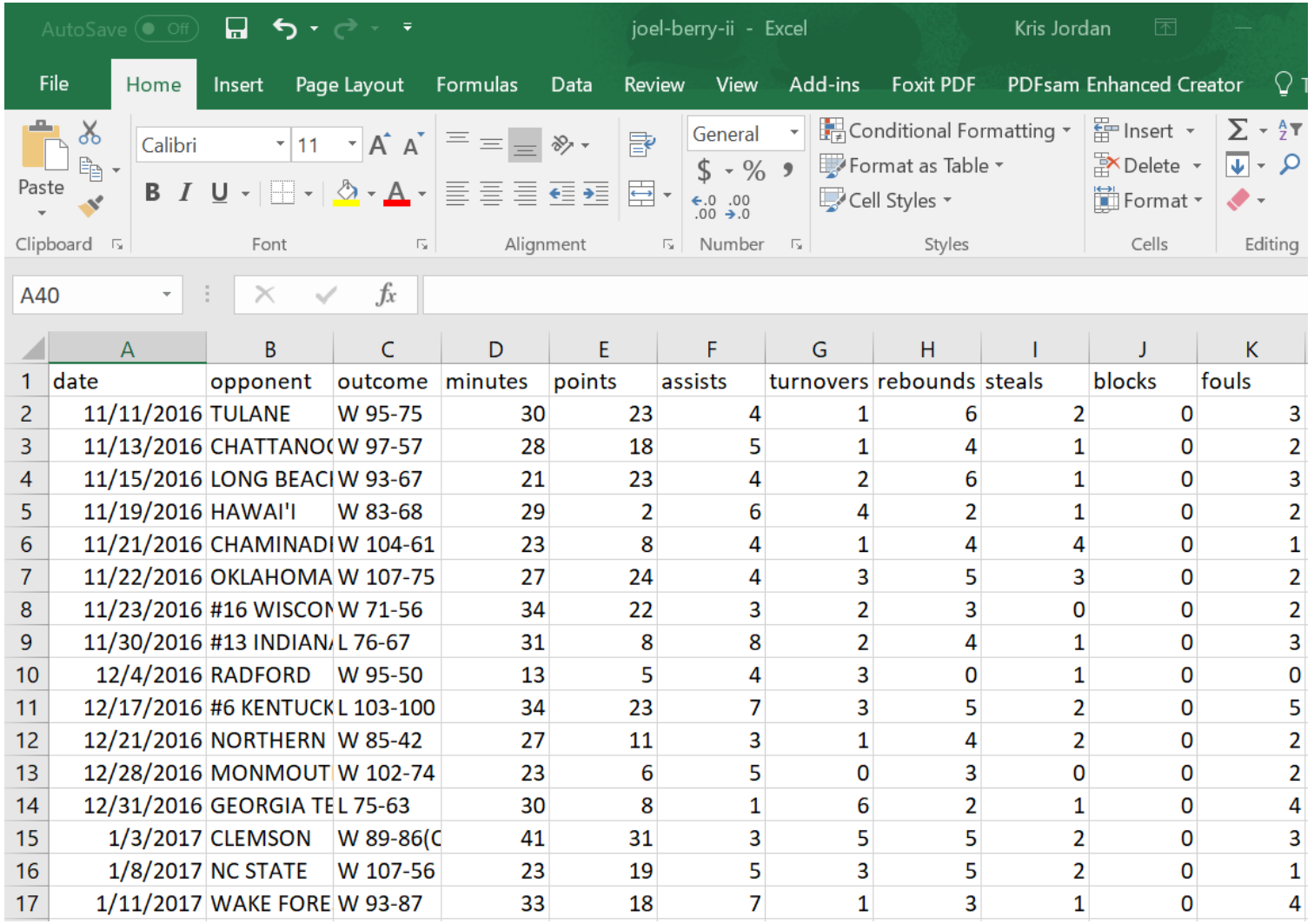
Season: 2016-17

2016-17 GAME LOG

DATE	OPP	RESULT	MIN	FGM-FGA	FG%	3PM-3PA	3P%	FTM-FTA	FT%	REB	AST	BLK	STL	PF	TO	PTS
Fri 11/11	@ TULANE	W 95-75	30	6-12	.500	4-7	.571	7-9	.778	6	4	0	2	3	1	23
Sun 11/13	vs CHATTANOOGA	W 97-57	28	5-8	.625	2-4	.500	6-6	1.000	4	5	0	1	2	1	18
Tue 11/15	vs LONG BEACH ST	W 93-67	21	7-10	.700	2-4	.500	7-7	1.000	6	4	0	1	3	2	23
Sat 11/19	@ HAWAI'I	W 83-68	29	1-9	.111	0-5	.000	0-0	.000	2	6	0	1	2	4	2
Mon 11/21	@ CHAMINADE	W 104-61	23	3-8	.375	2-5	.400	0-0	.000	4	4	0	4	1	1	8
Tue 11/22	vs OKLAHOMA STATE	W 107-75	27	7-10	.700	4-6	.667	6-6	1.000	5	4	0	3	2	3	24
Wed 11/23	vs #16 WISCONSIN	W 71-56	34	9-12	.750	2-3	.667	2-2	1.000	3	3	0	0	2	2	22
Wed 11/30	@ #13 INDIANA	L 76-67	31	3-13	.231	1-6	.167	1-2	.500	4	8	0	1	3	2	8
Sun 12/4	vs RADFORD	W 95-50	13	1-3	.333	1-3	.333	2-2	1.000	0	4	0	1	0	3	5

Today's Data

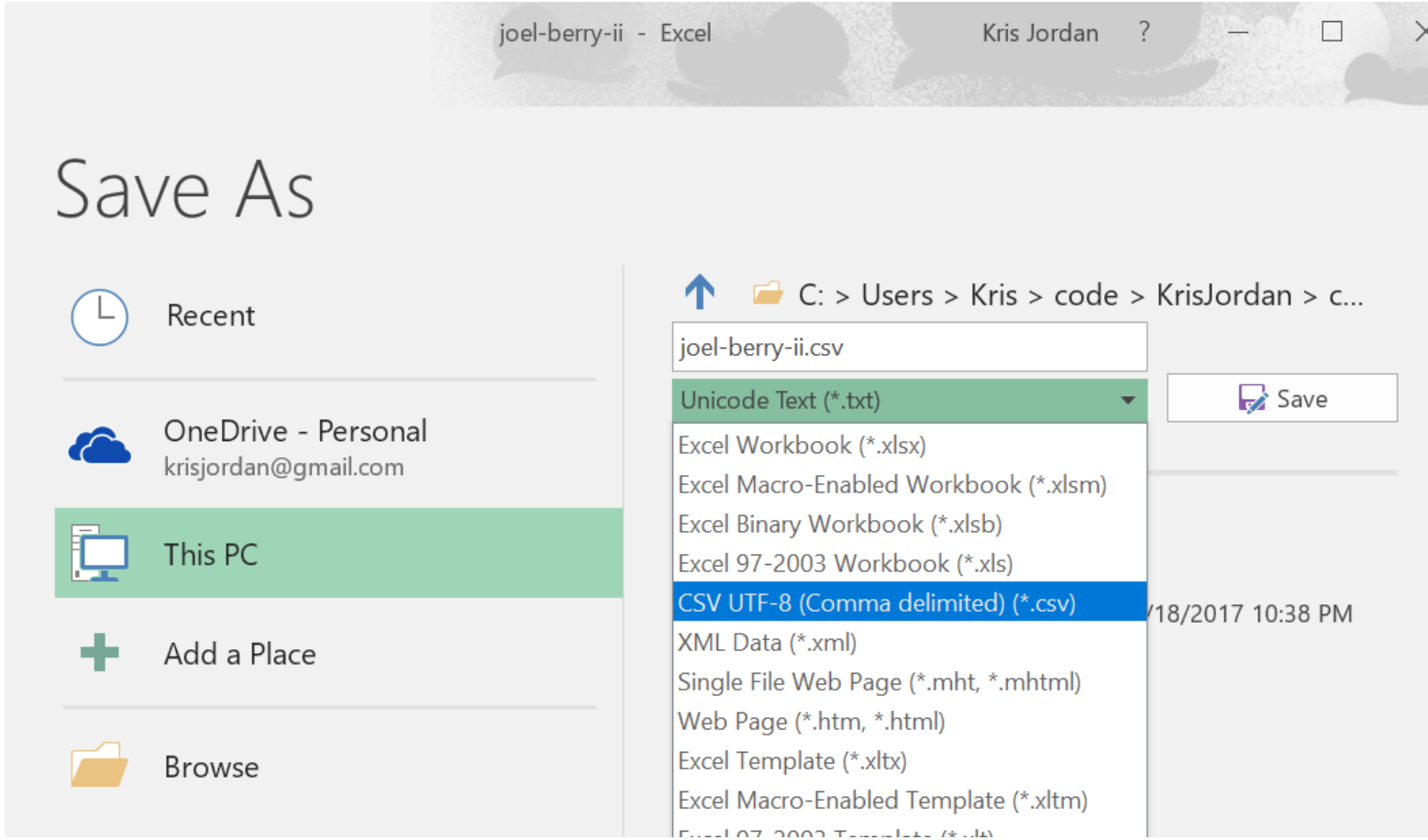
- The table was cleaned up a bit in Excel and formatting removed
- Column header names were changed to match properties we'll use in our code (we'll come back to this soon)



	A	B	C	D	E	F	G	H	I	J	K
1	date	opponent	outcome	minutes	points	assists	turnovers	rebounds	steals	blocks	fouls
2	11/11/2016	TULANE	W 95-75	30	23	4	1	6	2	0	3
3	11/13/2016	CHATTANOC	W 97-57	28	18	5	1	4	1	0	2
4	11/15/2016	LONG BEACI	W 93-67	21	23	4	2	6	1	0	3
5	11/19/2016	HAWAI'I	W 83-68	29	2	6	4	2	1	0	2
6	11/21/2016	CHAMINADI	W 104-61	23	8	4	1	4	4	0	1
7	11/22/2016	OKLAHOMA	W 107-75	27	24	4	3	5	3	0	2
8	11/23/2016	#16 WISCON	W 71-56	34	22	3	2	3	0	0	2
9	11/30/2016	#13 INDIAN/	L 76-67	31	8	8	2	4	1	0	3
10	12/4/2016	RADFORD	W 95-50	13	5	4	3	0	1	0	0
11	12/17/2016	#6 KENTUCK	L 103-100	34	23	7	3	5	2	0	5
12	12/21/2016	NORTHERN	W 85-42	27	11	3	1	4	2	0	2
13	12/28/2016	MONMOUT	W 102-74	23	6	5	0	3	0	0	2
14	12/31/2016	GEORGIA TE	L 75-63	30	8	1	6	2	1	0	4
15	1/3/2017	CLEMSON	W 89-86(C	41	31	3	5	5	2	0	3
16	1/8/2017	NC STATE	W 107-56	23	19	5	3	5	2	0	1
17	1/11/2017	WAKE FORE	W 93-87	33	18	7	1	3	1	0	4

Today's Data

- Finally it was saved as a special type of file:
- **CSV UTF-8 (Comma delimited) (*.csv)**
- This is a common data table format that is easy to work with in code.



Today's Data

- Here's what the contents of the CSV file look like.
- It is stored in:

data/joel-berry-ii.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.

joel-berry-ii.csv

```
date,opponent,outcome,minutes,points,assists,turnovers,rebounds,steals,blocks,fouls
2016-11-11,TULANE,W 95-75,30,23,4,1,6,2,0,3
2016-11-13,CHATTANOOGA,W 97-57,28,18,5,1,4,1,0,2
2016-11-15,LONG BEACH ST,W 93-67,21,23,4,2,6,1,0,3
2016-11-19,HAWAI'I,W 83-68,29,2,6,4,2,1,0,2
2016-11-21,CHAMINADE,W 104-61,23,8,4,1,4,4,0,1
2016-11-22,OKLAHOMA STATE,W 107-75,27,24,4,3,5,3,0,2
2016-11-23,#16 WISCONSIN,W 71-56,34,22,3,2,3,0,0,2
2016-11-30,#13 INDIANA,L 76-67,31,8,8,2,4,1,0,3
2016-12-04,RADFORD,W 95-50,13,5,4,3,0,1,0,0
2016-12-17,#6 KENTUCKY,L 103-100,34,23,7,3,5,2,0,5
2016-12-21,NORTHERN IOWA,W 85-42,27,11,3,1,4,2,0,2
2016-12-28,MONMOUTH,W 102-74,23,6,5,0,3,0,0,2
2016-12-31,GEORGIA TECH,L 75-63,30,8,1,6,2,1,0,4
2017-01-03,CLEMSON,W 89-86(OT),41,31,3,5,5,2,0,3
2017-01-08,NC STATE,W 107-56,23,19,5,3,5,2,0,1
2017-01-11,WAKE FOREST,W 93-87,33,18,7,1,3,1,0,4
2017-01-14,#9 FLORIDA STATE,W 96-83,35,26,1,2,2,2,0,3
2017-01-16,SYRACUSE,W 85-68,31,10,1,2,1,1,0,2
2017-01-21,BOSTON COLLEGE,W 90-82,35,9,0,2,0,1,1,1
2017-01-26,VIRGINIA TECH,W 91-72,30,15,4,0,3,1,1,2
2017-01-28,MIAMI,L 77-62,30,2,4,2,1,1,0,3
```

Modelling a "Game" with a class

- Each Game has properties associated with it:
  - date
  - opponent
  - points
  - and more...
- These are column names in our data table
- In our program, we'll declare a class to model a single Game's stats 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.

```
class Game {

   date: string = "";
   opponent: string = "";
   points: number = 0;
   fouls: number = 0;

}
```

How do we prompt the user for a CSV file?

- There's a function in the **introcs** library to do so!

- Documentation:

```
await csvToArray(prompt:string, cname:Class): Class[]
```

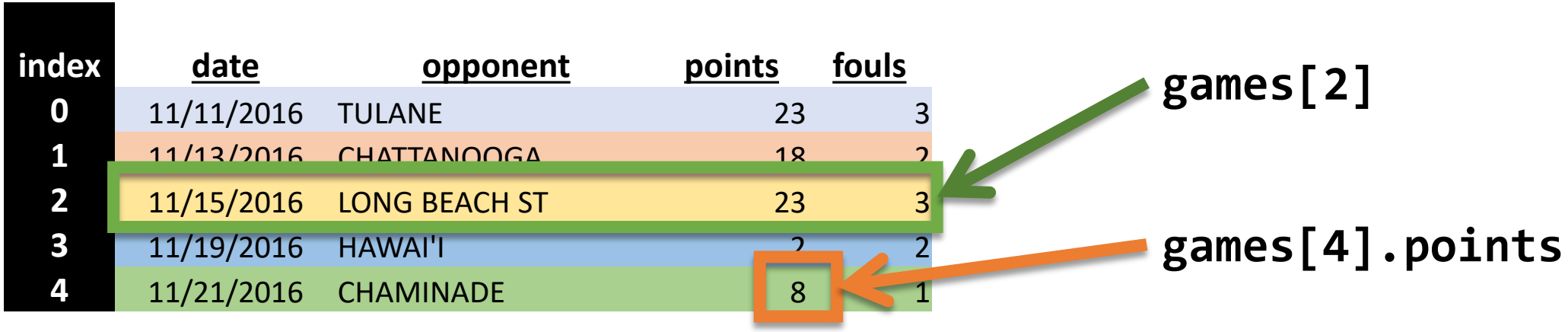
- Parameters:

1. **prompt**- a string value presented to the user as instructions
2. **cname**- the name of the class (i.e. **Game**) each row of the CSV corresponds to

Reading a CSV into an Array of **Game** Objects

- We are working with our data table as an "array of Games", i.e. **`Game[]`**

- Each row in the data table will have a Game object associated with it. Each column in the data table is a property of the Game object.



Hands-on: Filter 20 Point Games

- In the filter20Points function of 01-csv-data-app.ts:

1. Write a for loop to iterate through each index of the games array
2. In the for loop, establish a variable to hold the Game object at the current index being processed
3. If the current Game object's points property is at least 20 points, append the game object to the matches array

- Try calling this function and printing its return value from the main function.