Classes, Objects, and csv Processing

Lecture 10

Hands-on #0: Construct a Pizza Object

- Before you begin coding, open Pizza.ts
  - Talk with your neighbor about what is inside of this file

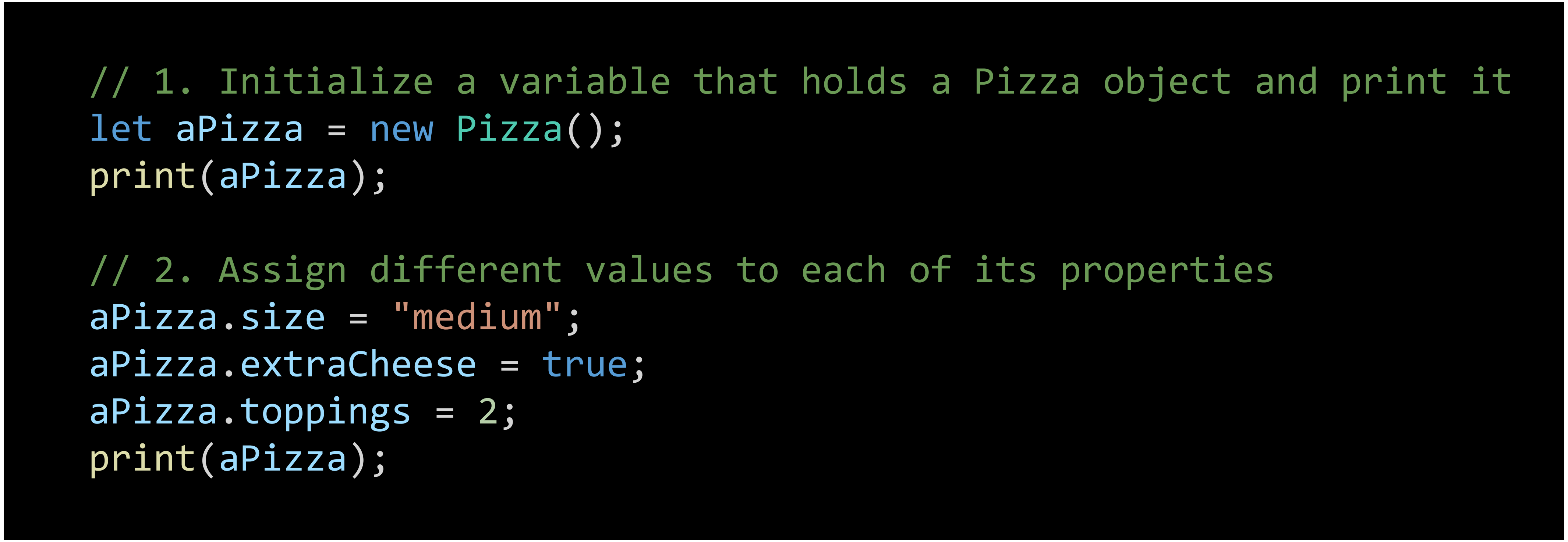
- In 00-pizza-price-app.ts
  - Notice the Pizza class is imported from "./Pizza"
  - Refer to your notes / video slides for specific syntax

  1. Declare a variable and assign it a Pizza object. Print this object.

  2. Assign different values to each of its three properties (size, extraCheese, toppings). After doing so, print the object again.

  3. (Ignore Todo #3)

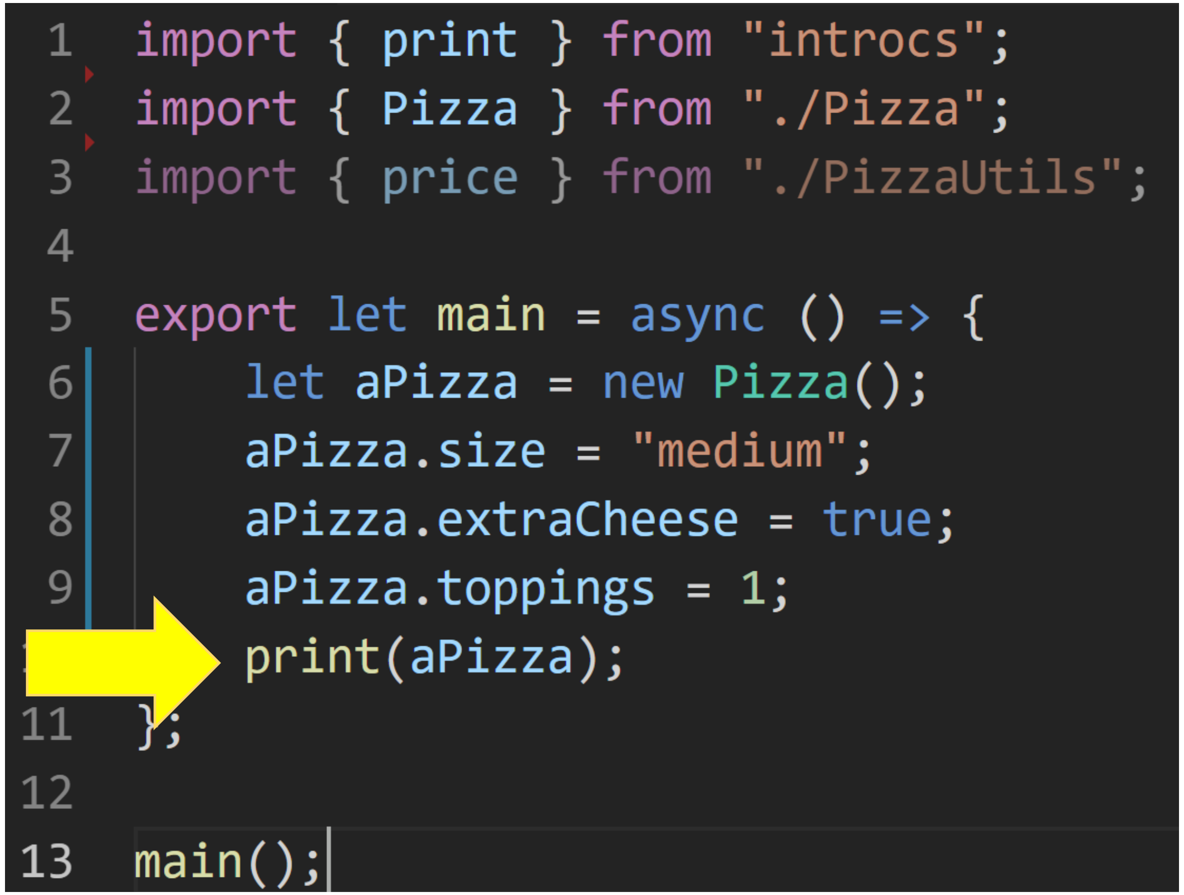
- Check-in on PollEv.com/compunc once complete!



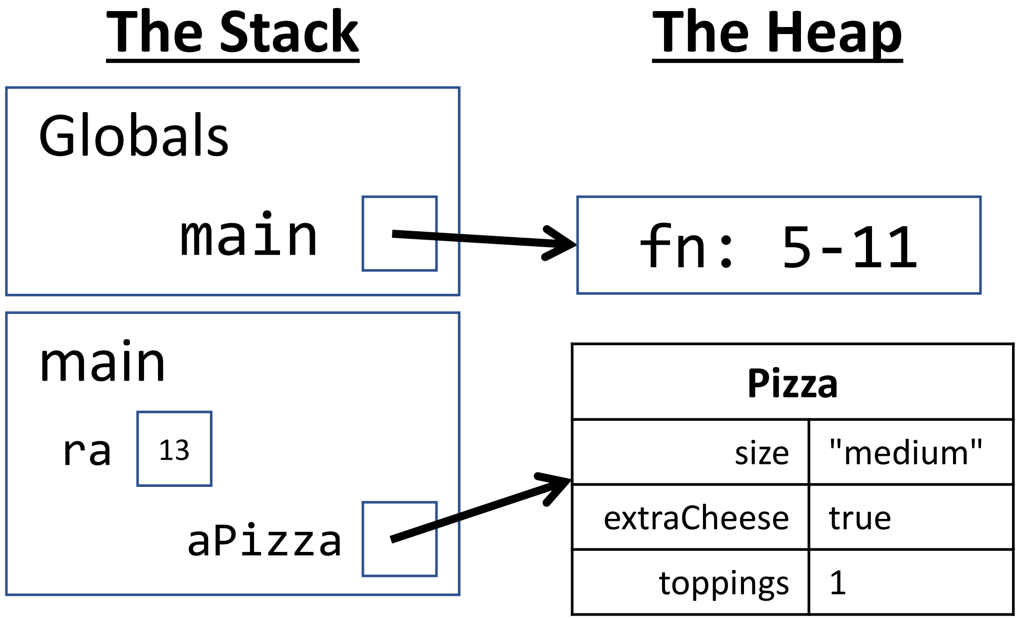
```
// 1. Initialize a variable that holds a Pizza object and print it
let aPizza = new Pizza();
print(aPizza);

// 2. Assign different values to each of its properties
aPizza.size = "medium";
aPizza.extraCheese = true;
aPizza.toppings = 2;
print(aPizza);
```

Object Values Live on the Heap

Like arrays, objects are *reference types*. Their variable names on the call stack hold references to their *actual values* in the heap.



```
import { print } from "introcs";
import { Pizza } from "./Pizza";
import { price } from "./PizzaUtils";

export let main = async () => {
    let aPizza = new Pizza();
    aPizza.size = "medium";
    aPizza.extraCheese = true;
    aPizza.toppings = 1;
    print(aPizza);
};

main();
```



Pizza	
size	"medium"
extraCheese	true
toppings	1

Be Careful to **Always Initialize your Variables**

Common Error:

```
Uncaught TypeError: Cannot set property '<property>' of undefined
```

- **Example:**

```
let pizza1: Pizza;
pizza1.size = "large"; // ERROR!!!
```

- **The fix:** `let pizza1 = new Pizza(); // Always initialize!`

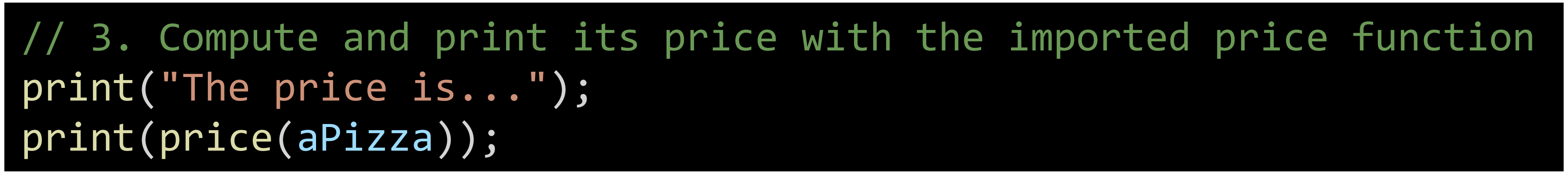
Hands-on #1: Calculate the Price of a Pizza

- Before you begin coding, open PizzaUtils.ts
  - Talk with your neighbor about what is inside of this file
- In 00-pizza-price-app.ts
  - Notice the `price` function is imported from "./PizzaUtils"

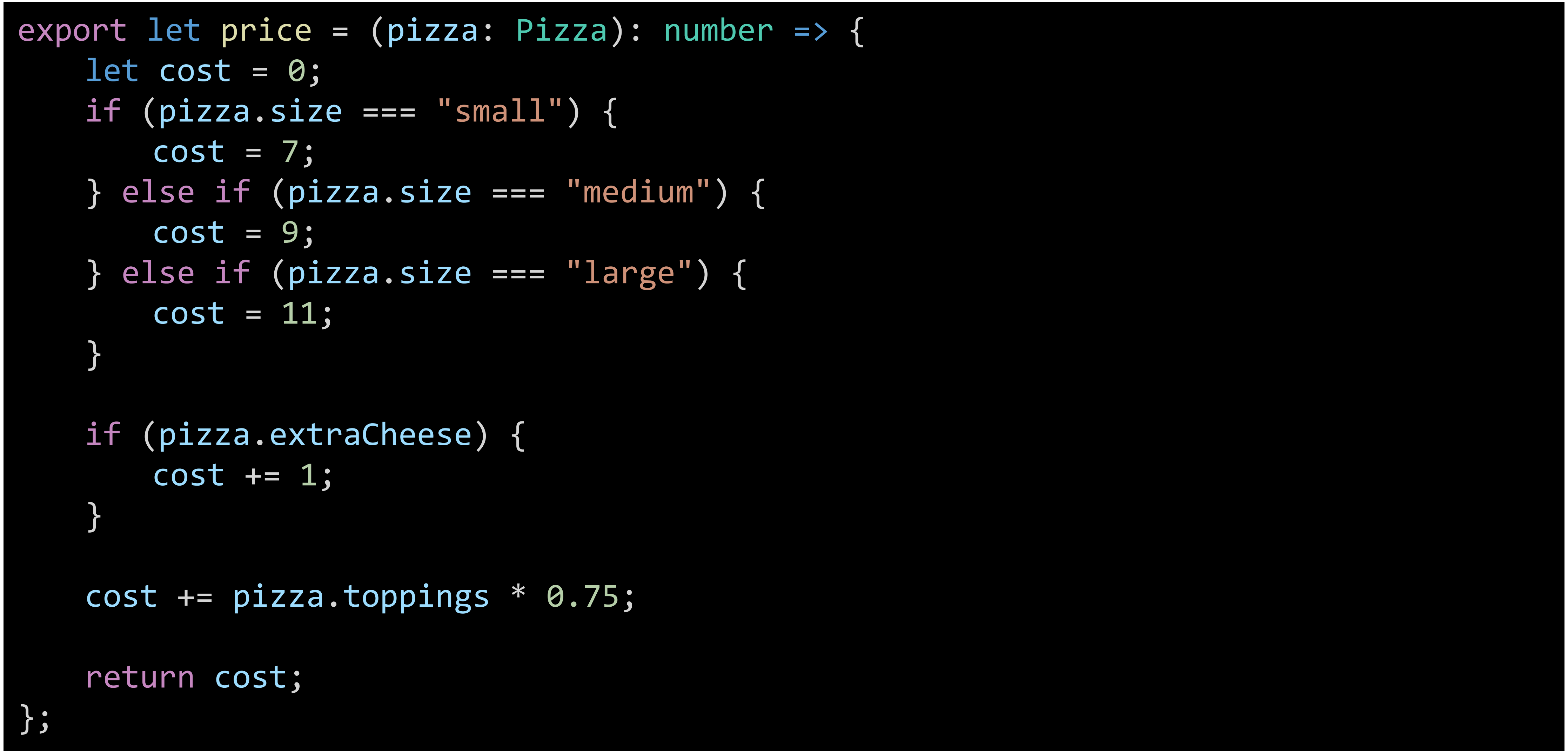
3. Call the `price` function with your Pizza object and print the return value. It should print 0 at this point because price is a skeleton function.

4. Correctly implement the `price` function in **PizzaUtils.ts**:
   - Size sets a base price of $7 small, $9 medium, $11 large
   - Extra cheese adds $1
   - Each topping costs $0.75

- Check-in on PollEv.com/compunc once your pizza price is correctly calculating! Try changing property values to inspect.



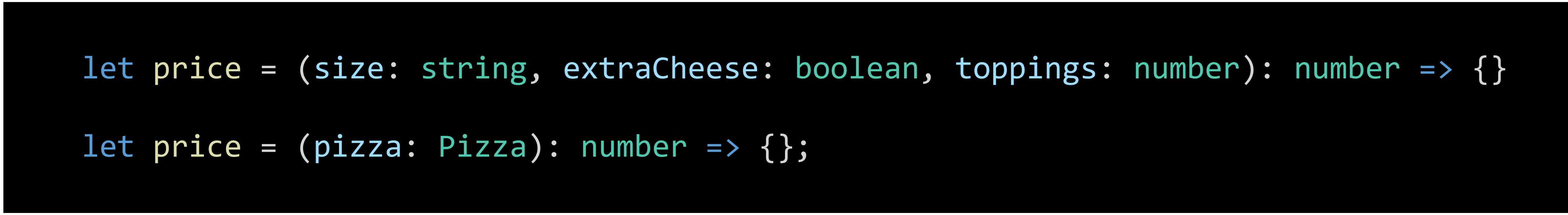
```
// 3. Compute and print its price with the imported price function
print("The price is...");
print(price(aPizza));
```



```
export let price = (pizza: Pizza): number => {
    let cost = 0;
    if (pizza.size === "small") {
        cost = 7;
    } else if (pizza.size === "medium") {
        cost = 9;
    } else if (pizza.size === "large") {
        cost = 11;
    }

    if (pizza.extraCheese) {
        cost += 1;
    }

    cost += pizza.toppings * 0.75;

    return cost;
};
```

The "Bundling" of Related Values is an Important Benefit of Composite Data Types / Objects

- Consider the following two function signatures...



```
let price = (size: string, extraCheese: boolean, toppings: number): number => {}

let price = (pizza: Pizza): number => {};
```

- Notice with a Pizza data type the function's *semantics* are improved
  - Is the first function calculating the price of a cheese burger?
  - The second function's signature reads more meaningfully...
    "`price` is a function that is given a `Pizza` object and returns a `number`"

- Consider an object with *far more* properties...
  - Pizza: Base sauce, gluten free crust, thin vs. deep dish, ...
  - Objects give us a convenient means for tightly packaging related variables together

Arrays of Objects

- You can make an array of objects!

  Declaration is just the same...

```
let <arrayName>: <type>[] = [];
ex: let orders: Pizza[] = [];
```

- Initializing an element requires constructing an object:

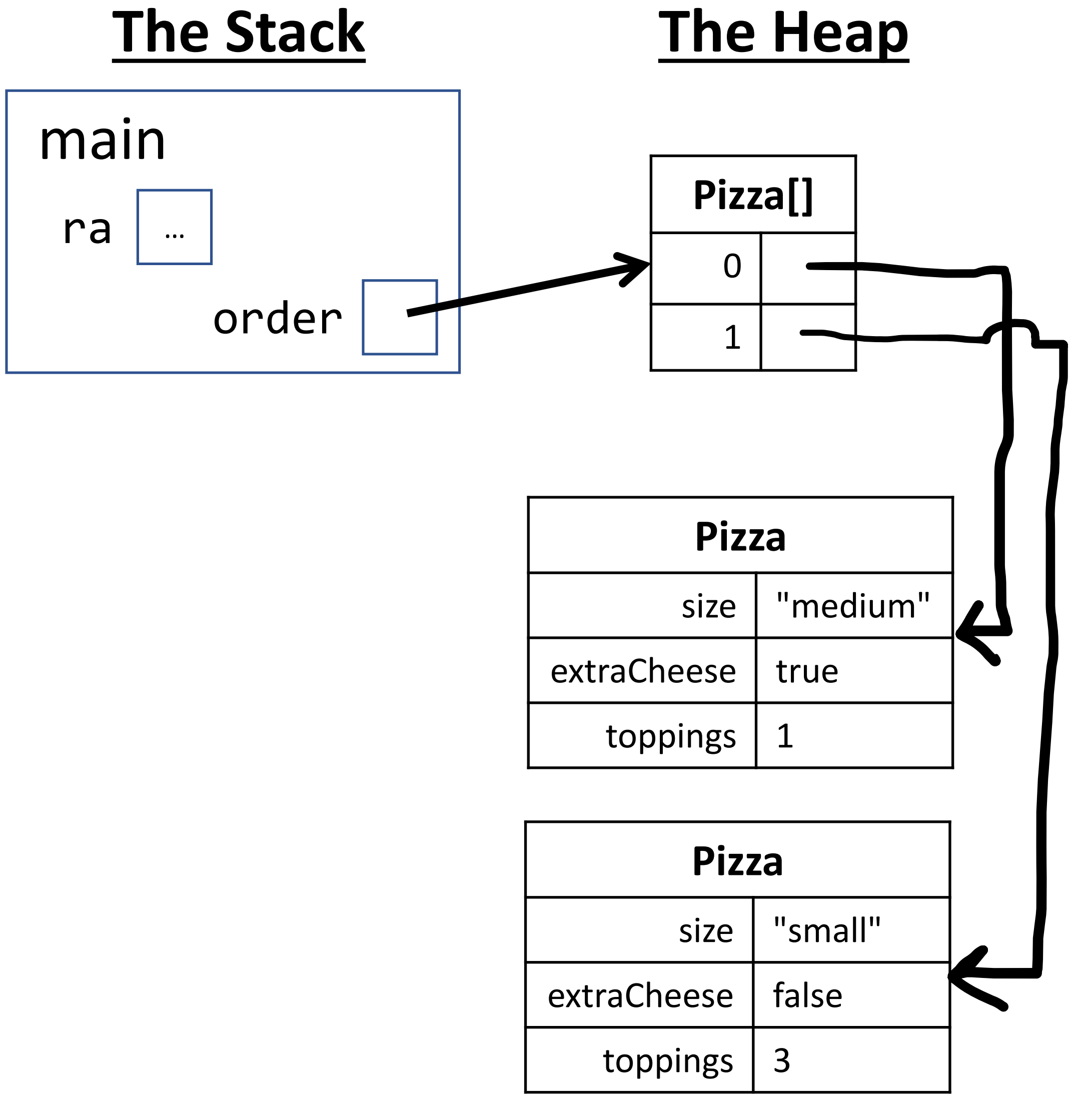
```
<arrayName>[<index>] = new <type>();
ex: orders[0] = new Pizza();
```

- Accessing an element is also the same:

```
<arrayName>[<index>]
ex: orders[1]
```

- To access a property, use the dot operator:

```
<arrayName>[<index>].<propertyName>
ex: orders[1].toppings
```



The Stack

main

ra ...

order

The Heap

Pizza[]	
0	
1	

Pizza	
size	"medium"
extraCheese	true
toppings	1

Pizza	
size	"small"
extraCheese	false
toppings	3

Follow Along: Working with Arrays of Objects

- Open 01-pizza-order-app.ts
- Notice that the **order** variable's type is a **Pizza[]**

- After the while loop completes:

1. Print the **order** array
2. Print the first element of the **order** array
3. Print the **size** property of the first element of the **order** array

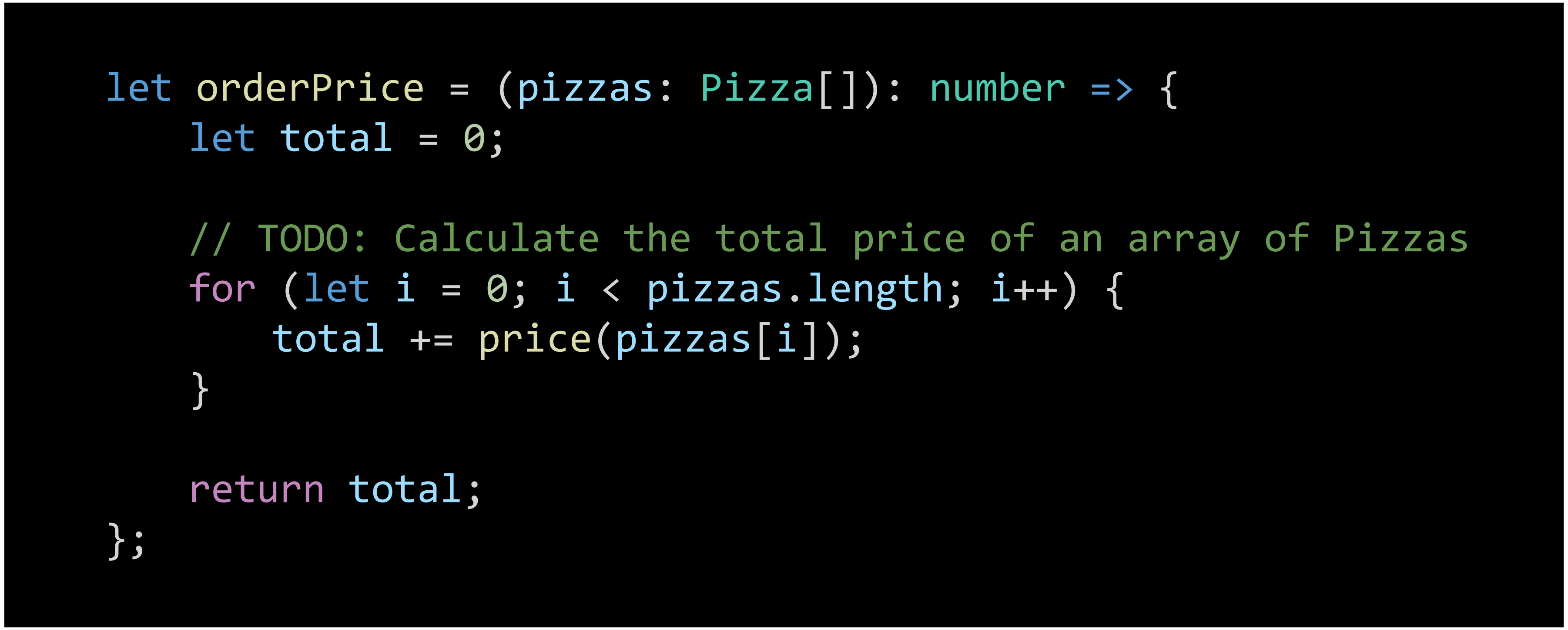
```
print("The order is...");
// TODO 1: Print the order
print(order);

print("The first Pizza is...");
// TODO 2: Print the 1st pizza at index 0
print(order[0]);

print("The first Pizza's toppings are...");
// TODO 3: Print the 1st pizza's toppings
print(order[0].toppings);
```

Hands-on: Iterating over an Array of Objects

- In 01-pizza-order-app.ts
- In the **`main`** function, call the **`orderPrice`** function and print its return value.
- Then, correctly implement the **`orderPrice`** function skeleton:

1. Loop over each of the Pizza objects in the pizzas parameter
2. Call the **`price`** function (imported) with each pizza
3. Add the price of each pizza to the total

- Check-in when you're calculating the total price of an array of Pizzas.



```
let orderPrice = (pizzas: Pizza[]): number => {
    let total = 0;

    // TODO: Calculate the total price of an array of Pizzas
    for (let i = 0; i < pizzas.length; i++) {
        total += price(pizzas[i]);
    }

    return total;
};
```

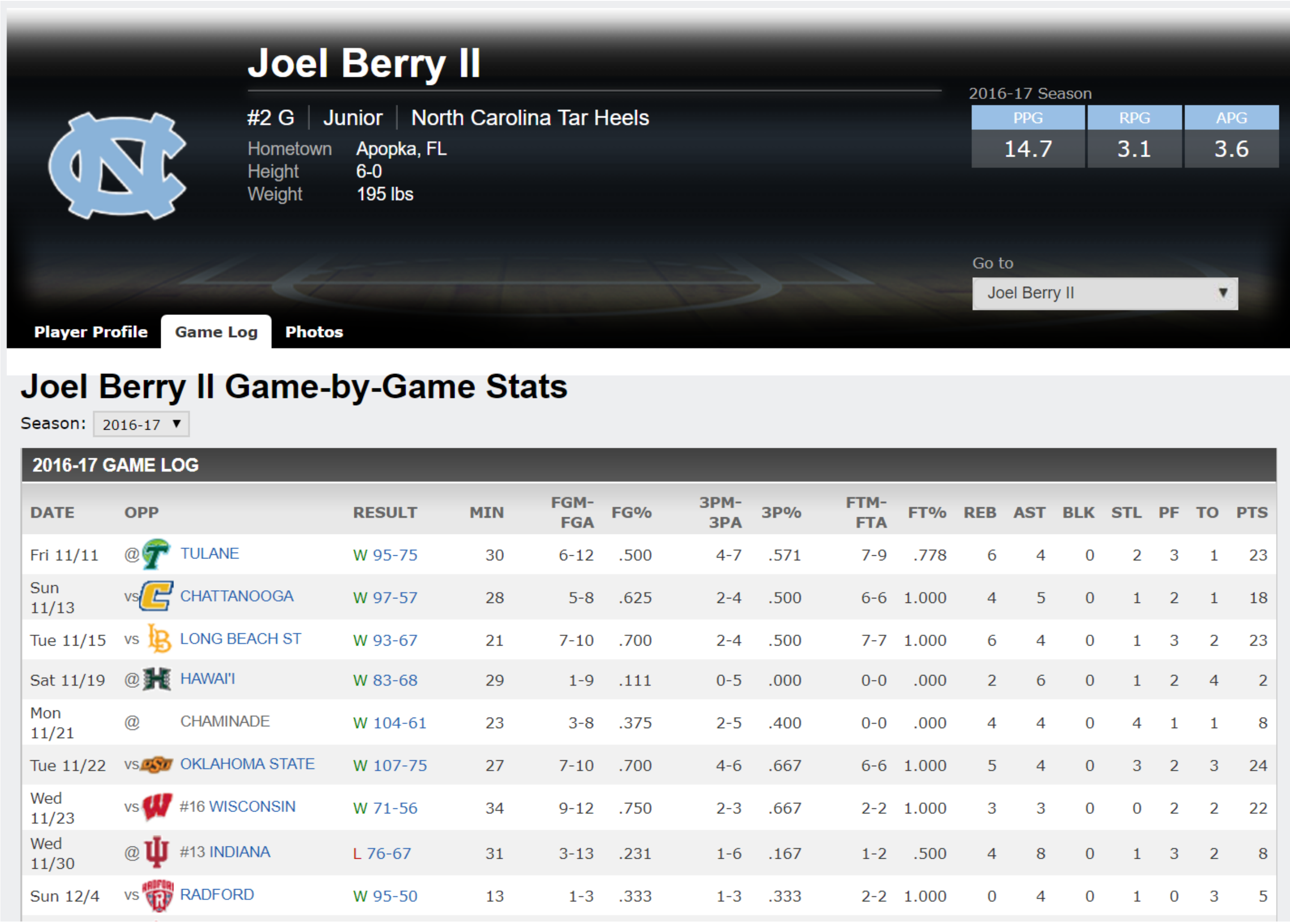
Working with Data

- **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

#2 G | Junior | North Carolina Tar Heels

Hometown Apopka, FL
Height 6-0
Weight 195 lbs

2016-17 Season

PPG	RPG	APG
14.7	3.1	3.6

Go to
Joel Berry II ▼

Player Profile | Game Log | Photos

Joel Berry II Game-by-Game Stats

Season: 2016-17 ▼

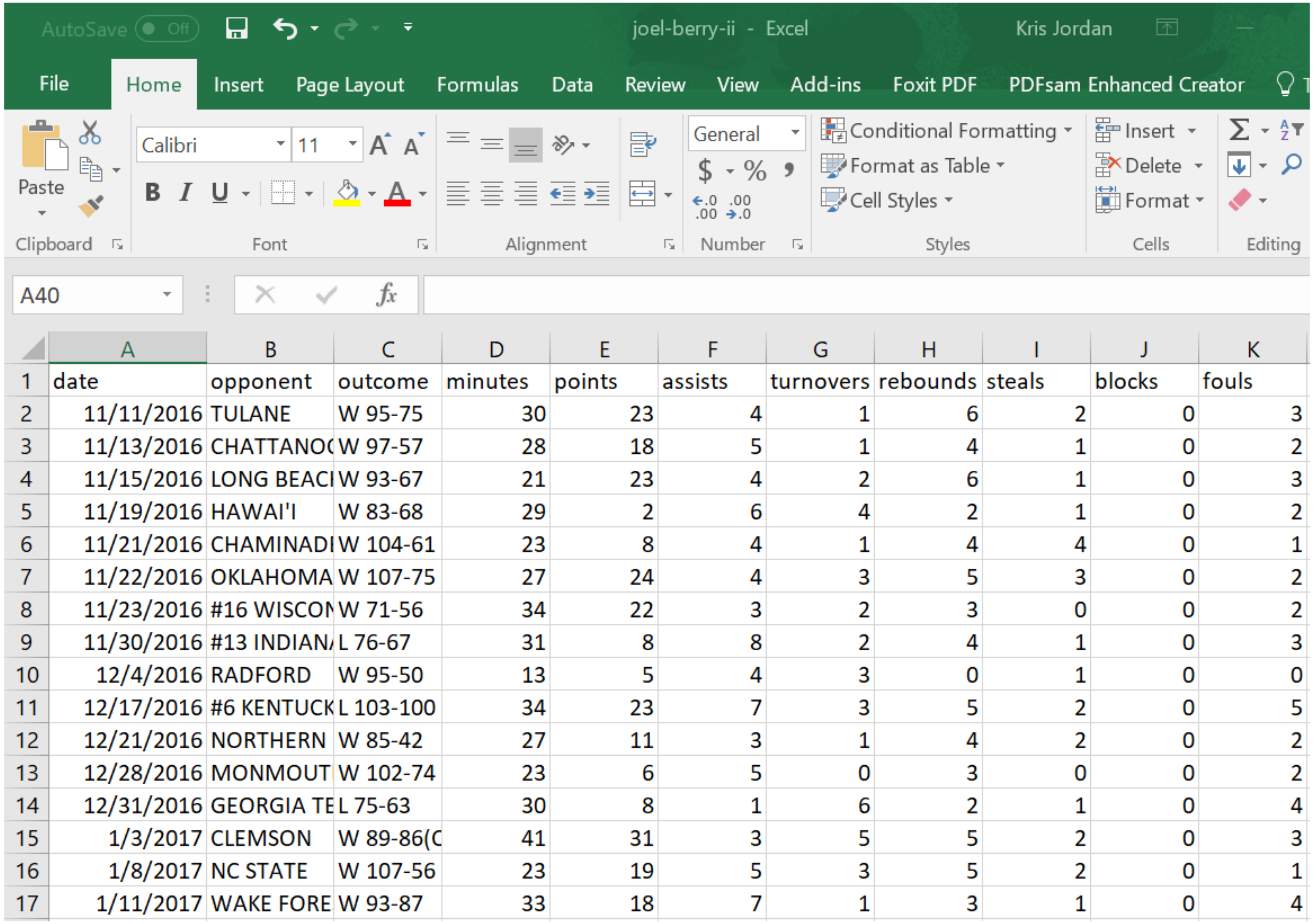
2016-17 GAME LOG

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

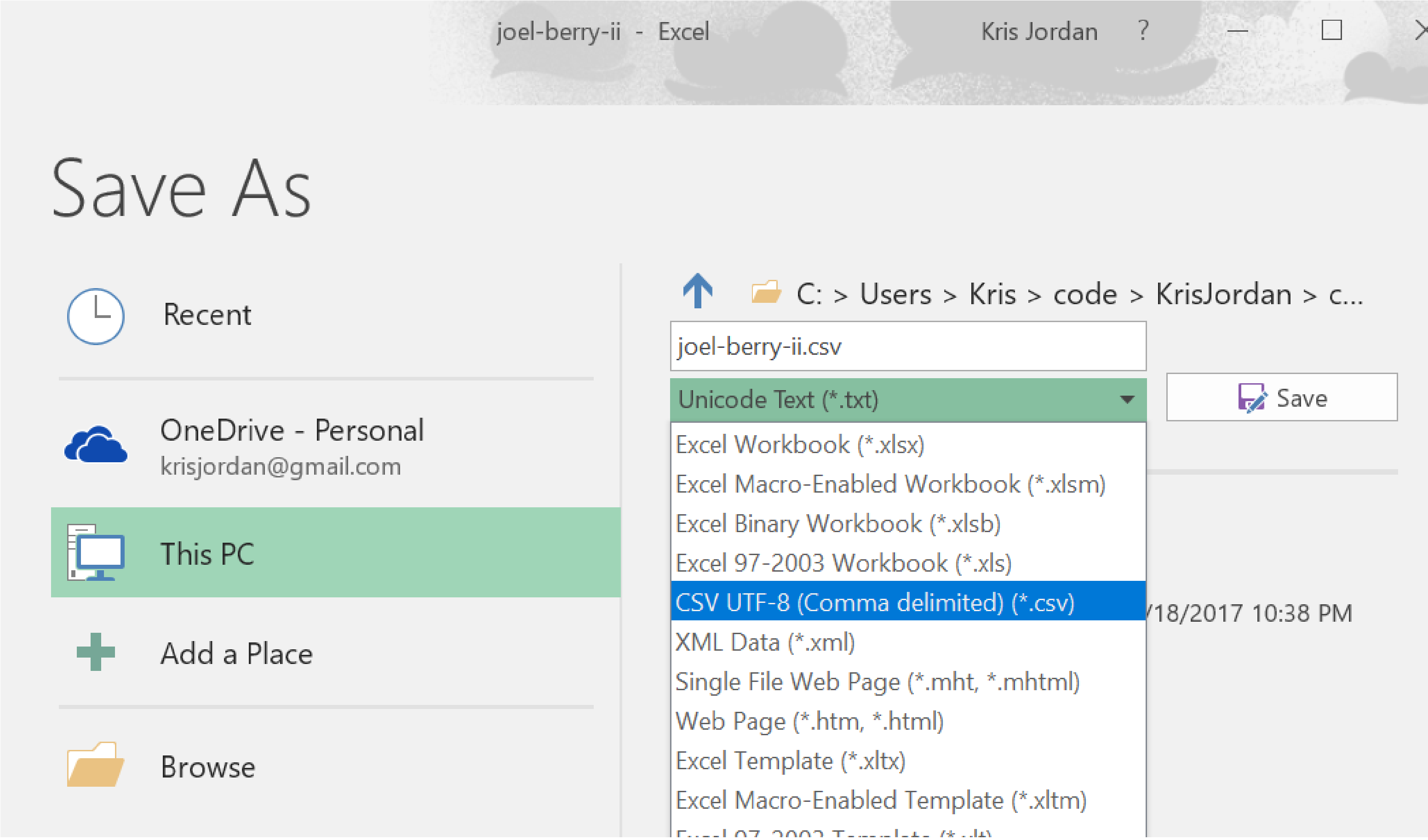
- Column header names were changed to match properties we'll use in our code (we'll come back to this soon)



	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	CHATTANO(	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.



joel-berry-ii.csv

Unicode Text (*.txt)

Excel Workbook (*.xlsx)
Excel Macro-Enabled Workbook (*.xlsm)
Excel Binary Workbook (*.xlsb)
Excel 97-2003 Workbook (*.xls)
CSV UTF-8 (Comma delimited) (*.csv)
XML Data (*.xml)
Single File Web Page (*.mht, *.mhtml)
Web Page (*.htm, *.html)
Excel Template (*.xltx)
Excel Macro-Enabled Template (*.xltm)
Excel 97-2003 Template (*.xlt)

Today's Data

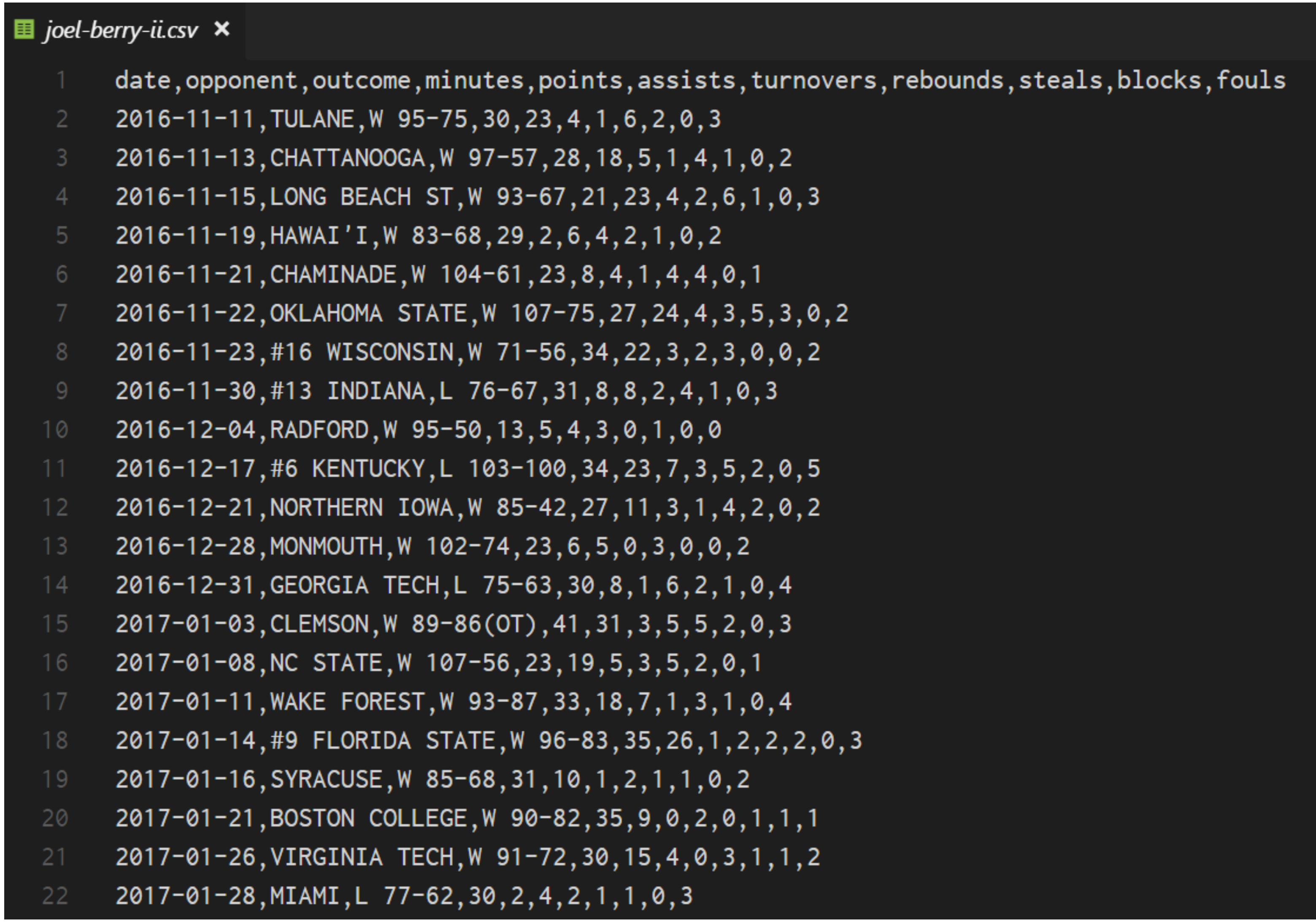
- Here's what the contents of the CSV file look like.

- It is stored in:

data/joel-berry-ii.csv

- Notice it's just plain text!

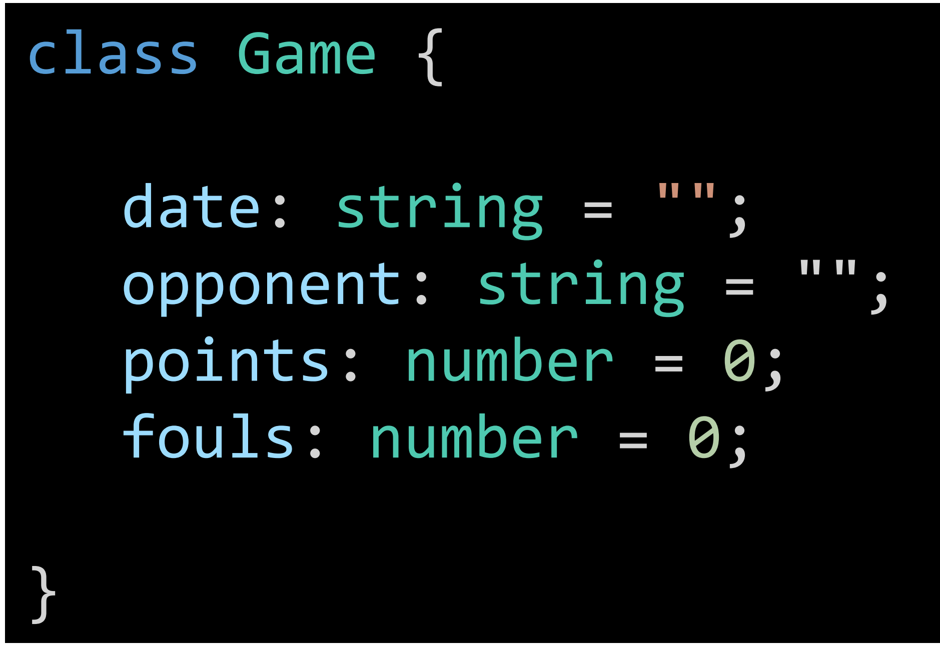
- Each row gets a line, each column is separated by a comma, hence "Comma Separated Values (CSV)" file.



```
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[]
```

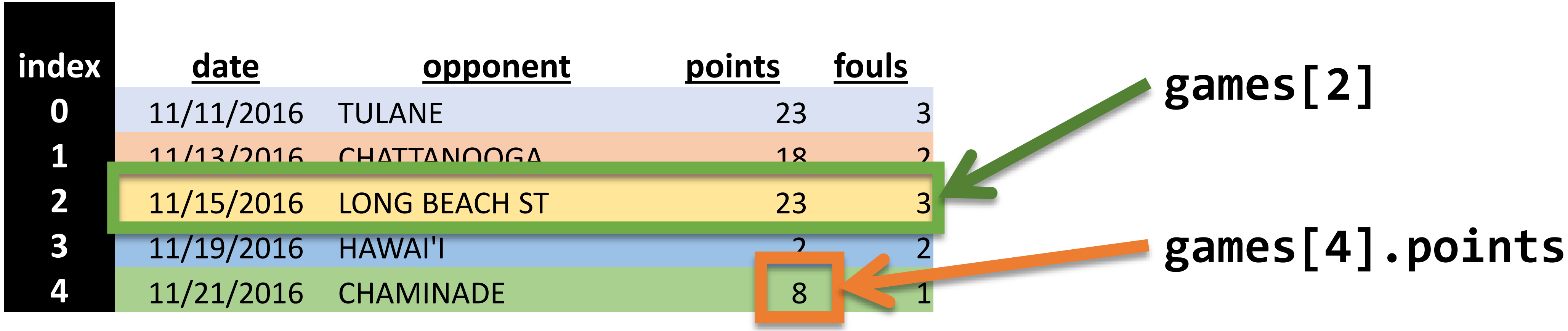
- Parameters:

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.



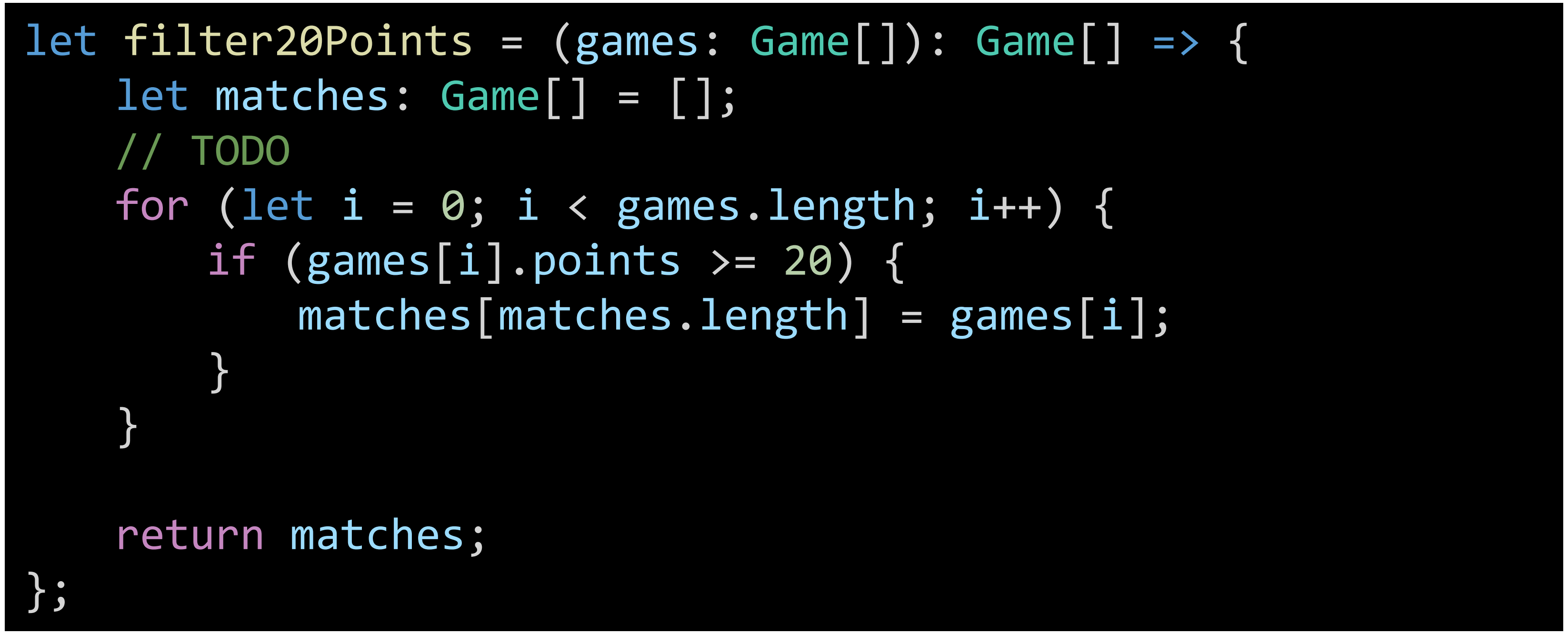
index	date	opponent	points	fouls
0	11/11/2016	TULANE	23	3
1	11/13/2016	CHATTANOOGA	18	2
2	11/15/2016	LONG BEACH ST	23	3
3	11/19/2016	HAWAI'I	2	2
4	11/21/2016	CHAMINADE	8	1

`games[2]`

`games[4].points`

Follow-Along: Filtering to 20 Point Games

- Let's write a function that *filters* an array of Games and returns an array of Games where every Game has 20 or more points.



```
let filter20Points = (games: Game[]): Game[] => {
    let matches: Game[] = [];
    // TODO
    for (let i = 0; i < games.length; i++) {
        if (games[i].points >= 20) {
            matches[matches.length] = games[i];
        }
    }

    return matches;
};
```