IN4MATX 285: Interactive Technology Studio

Programming: Data Structures and Functions in JavaScript

Today’s goals

By the end of today, you should be able to...

- Use arrays in Javascript to list items
- Use objects in Javascript to organize content by keys and values
- Manipulate arrays and objects to add new values and change existing ones
- Loop over arrays and dictionaries to access data
- Create and call functions to separate out code

So imagine, we have a grocery list…

Grocery list

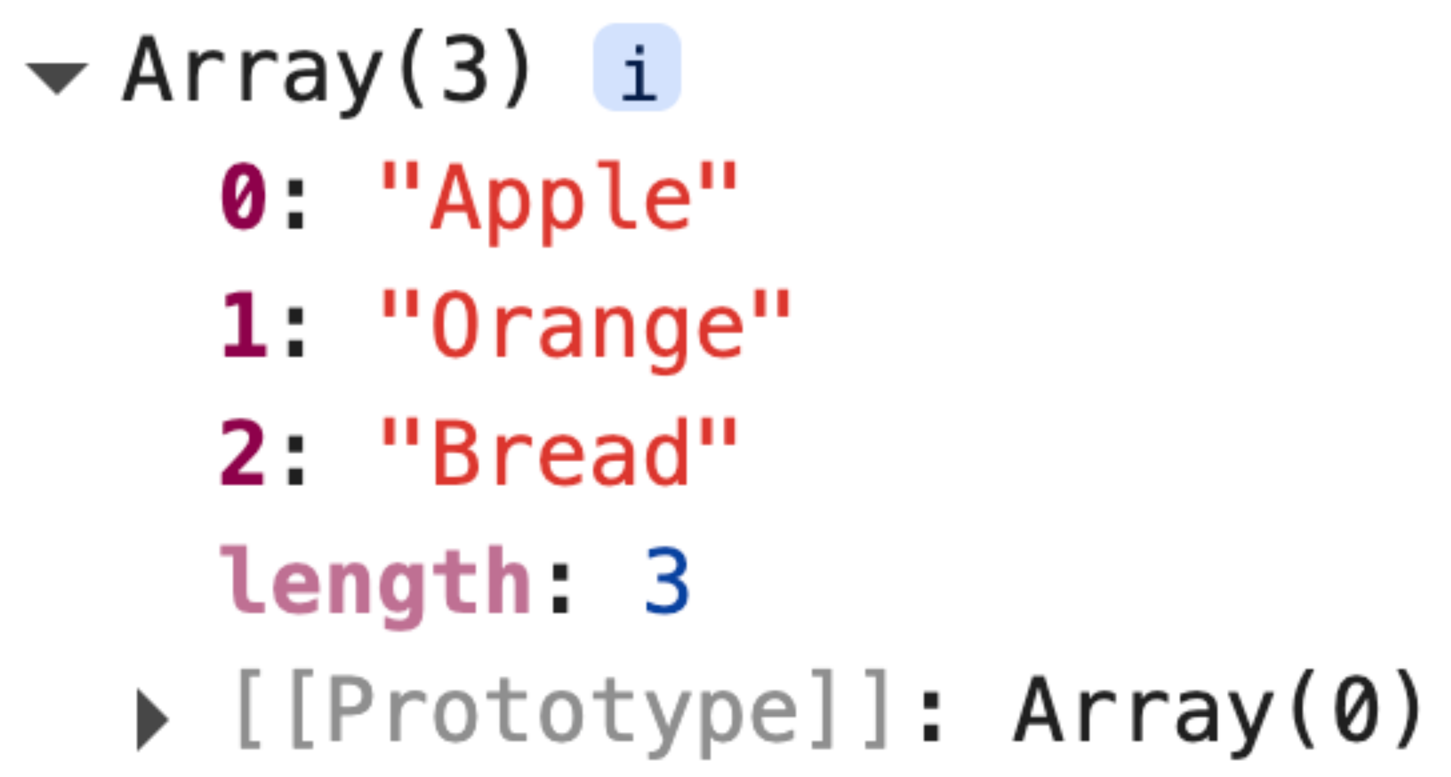
```
let groceryItem1 = 'Apple';
let groceryItem2 = 'Orange';
let groceryItem3 = 'Bread';
```

- What if there were **50** items in my grocery list? **5,000**?
- Solution: arrays

Arrays

- Arrays: a new type of variable

```
let groceryItems = ['Apple', 'Orange', 'Bread'];
console.log(groceryItems);
```

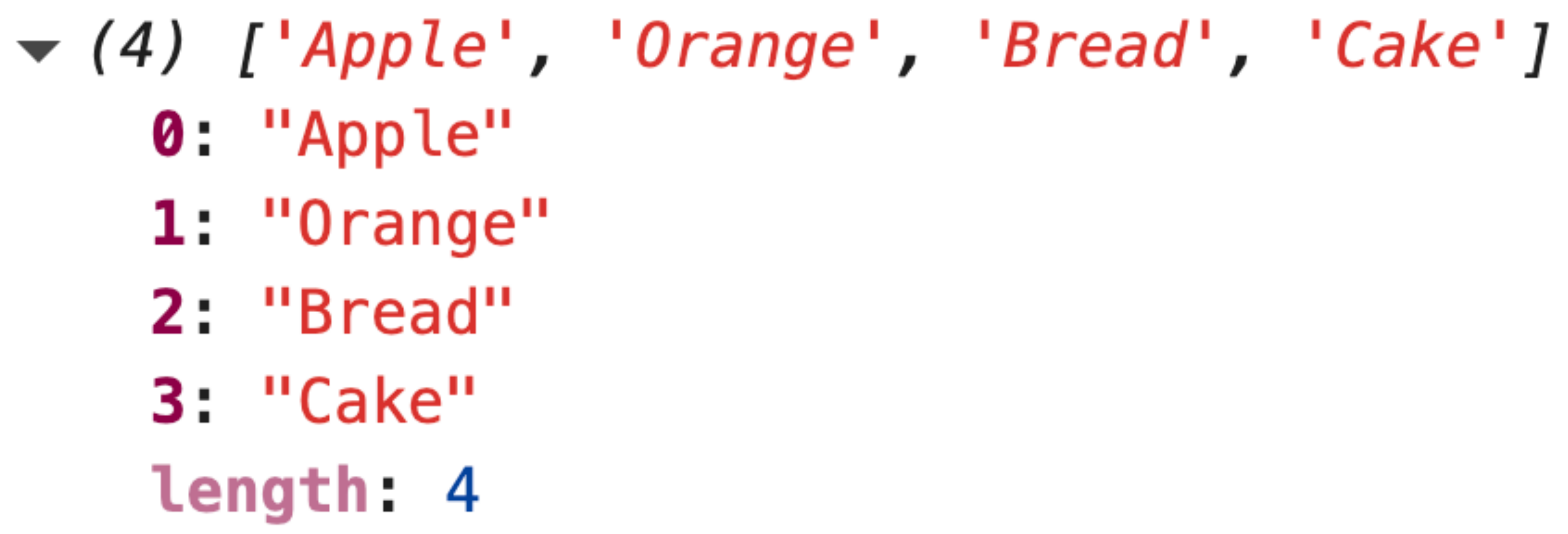


```
▾ Array(3) i
    0: "Apple"
    1: "Orange"
    2: "Bread"
    length: 3
  ▸ [[Prototype]]: Array(0)
```

Arrays

- You can add to arrays with `push`!

```
let groceryItems = ['Apple', 'Orange', ‘Bread'];
groceryItems.push('Cake');
console.log(groceryItems);
```



```
▼ (4) ['Apple', 'Orange', 'Bread', 'Cake']
    0: "Apple"
    1: "Orange"
    2: "Bread"
    3: "Cake"
    length: 4
```

Arrays

- Arrays can be *accessed*

- Counting starts at 0

```
let groceryItems = ['Apple', 'Orange', 'Bread'];
console.log(groceryItems[1]); Orange
```

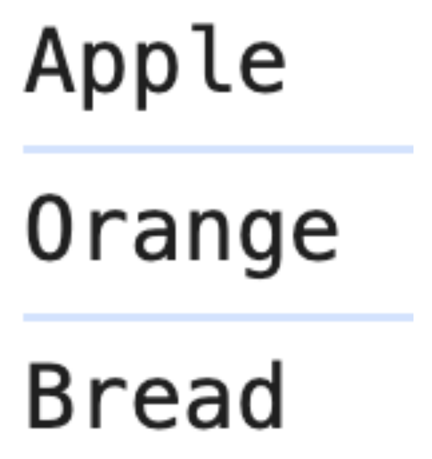
- Arrays have a *length* property

```
console.log(groceryItems.length); 3
```

Arrays

- With accessing and the length, you can *loop* over arrays

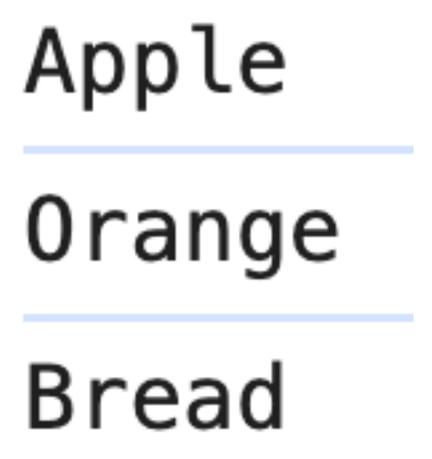
```
let groceryItems = ['Apple', 'Orange', 'Bread'];
for(let i=0; i < groceryItems.length; i = i + 1) {
    console.log(groceryItems[i]);
}
```



```
Apple
Orange
Bread
```

- The `of` keyword allow for looping over all items in an array

```
for(let groceryItem of groceryItems) {
    console.log(groceryItem);
}
```



```
Apple
Orange
Bread
```

Arrays

- Arrays can be of any type, or even a mix of types

```
let letters = ['a', 'b', 'c'];
let numbers = [1, 2, 3];
let things = [‘abc', 2.5, true, [5, 9, 8]]; //arrays can be nested
let empty = [];

//access using [] notation
console.log( letters[1] ); //=> "b"
console.log( things[3][2] ); //=> 8

//assign using [] notation
letters[0] = 'z';
console.log( letters ); //=> ['z', 'b', 'c']
```

Now, what if we wanted to associate prices with our grocery items? Number to purchase?

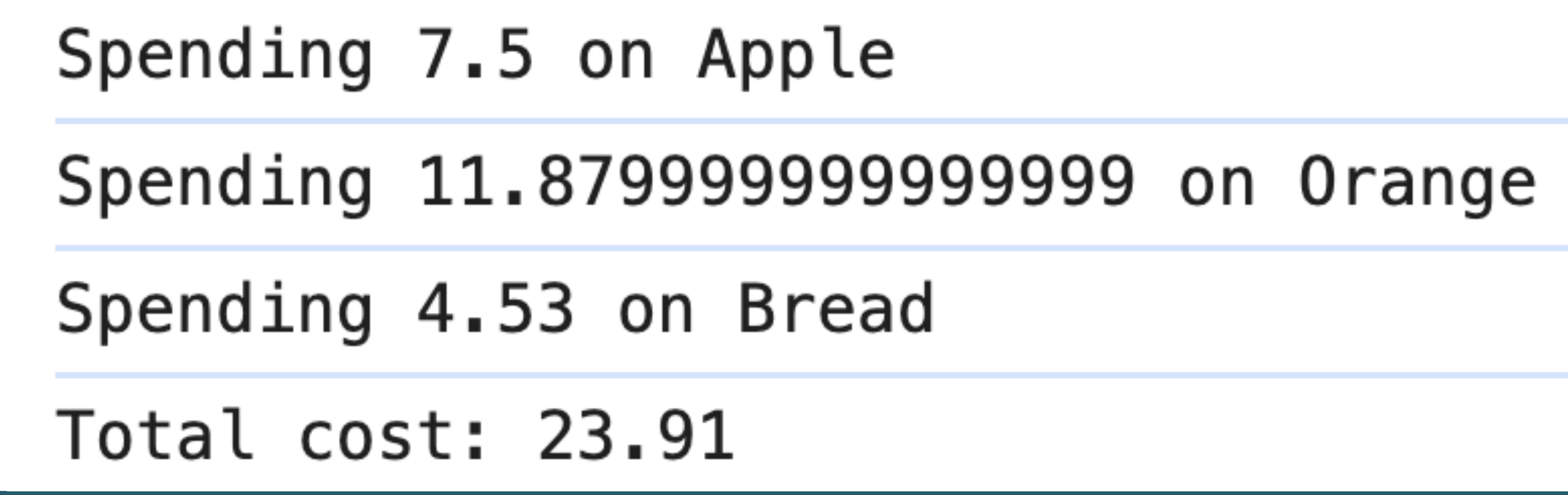
Multiple arrays

```
let groceryItems = ['Apple', 'Orange', 'Bread'];
let prices = [1.25, 0.99, 4.53];
let numberToPurchase = [6, 12, 1];

let totalCost = 0;

for(let i=0; i < groceryItems.length; i = i + 1) {
    let costOnItem = prices[i] * numberToPurchase[i];
    console.log('Spending ' + costOnItem + ' on ' + groceryItems[i]);
    totalCost = totalCost + costOnItem;
}

console.log('Total cost: ' + totalCost);
```



```
Spending 7.5 on Apple
Spending 11.879999999999999 on Orange
Spending 4.53 on Bread
Total cost: 23.91
```

Multiple arrays

- But, these arrays aren't really *associated* with one another
  - Nothing preventing you from adding more items, but not defining their price

```
let groceryItems = ['Apple', 'Orange', 'Bread'];
let prices = [1.25, 0.99, 4.53];
let numberToPurchase = [6, 12, 1];
```

- How might we associate item with price and amount?

Associative arrays/Objects

Associative arrays/Objects

- Associative arrays are Objects. They're another type of variable
- Objects allow for storing many values, as name:value pairs

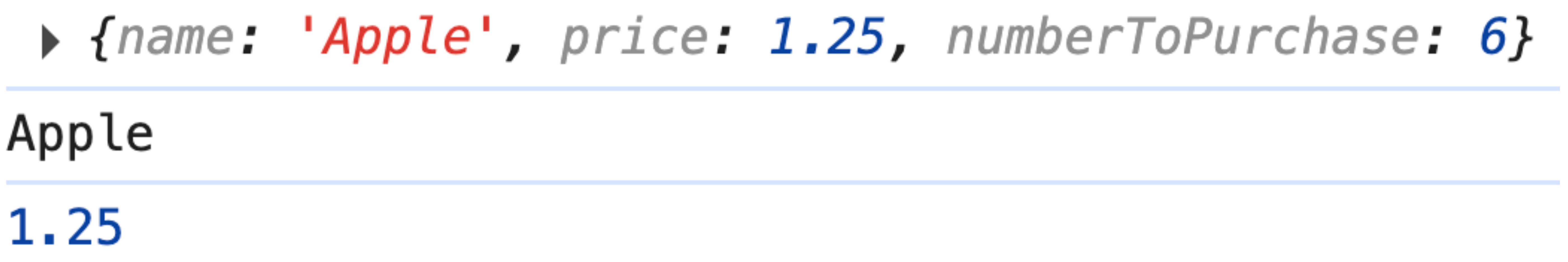
```
let groceryItem = {'name':'Apple', 'price': 1.25,
'numberToPurchase': 6};
```

Associative arrays/Objects

```
let groceryItem = {'name':'Apple', 'price': 1.25,
'numberToPurchase': 6};
```

- Objects can be *accessed*

```
console.log(groceryItem);
console.log(groceryItem['name']);
console.log(groceryItem.price);//Works the same as as
above
```



```
▸ {name: 'Apple', price: 1.25, numberToPurchase: 6}
Apple
1.25
```

- Objects can be *assigned* to

```
groceryItem['type'] = 'Granny Smith';
```

Objects

- Objects have no order
- Names are strings, values can be any type

Quotes around names are optional

```
ages = {alice:40, bob:35, charles:13}
extensions = {'daniel':1622, 'in4matx':9937}
num_words = {1:'one', 2:'two', 3:'three'}
things = {num:12, dog:'woof', list:[1,2,3]}
empty = {}
empty = new Object(); //empty object
```

JavaScript Object Notation (JSON)

```
{
  "first_name": "Alice",
  "last_name": "Smith",
  "age": 40,
  "pets": ["rover", "fluffy", "mittens"],
  "favorites": {
    "music": "jazz",
    "food": "pizza",
    "numbers": [12, 42]
  }
}
```

- Used in many APIs to send/receive data

Accessing properties

- Values can also be referenced with dot notation

```
var person = {
  firstName: 'Alice',
  lastName: 'Smith',
  favorites: {
    food: 'pizza',
    numbers: [12, 42]
  }
};

var name = person.firstName; //get value of 'firstName' key
person.lastName = 'Jones'; //set value of 'lastName' key
console.log(person.firstName+' '+person.lastName); //"Alice Jones"

var topic = 'food'
var favFood = person.favorites.food; //object in the object
              //object         //value

var firstNumber = person.favorites.numbers[0]; //12
person.favorites.numbers.push(7); //push 7 onto the Array
```

Useful object methods

- `Object.keys`
  - returns an array containing the keys
  - order is not guaranteed
  - Or `Object.values(`object`)` to get an array of the values
  - Or `Object.entries(`object`)` to get an array containing an array of key, value pairs

```
obj = { pet1: 'Dog', pet2: 'Cat' };

console.log(Object.entries(obj));
// [ ["pet1", "Dog"], ["pet2", "Cat"] ]
```

https://codeburst.io/useful-javascript-array-and-object-methods-6c7971d93230

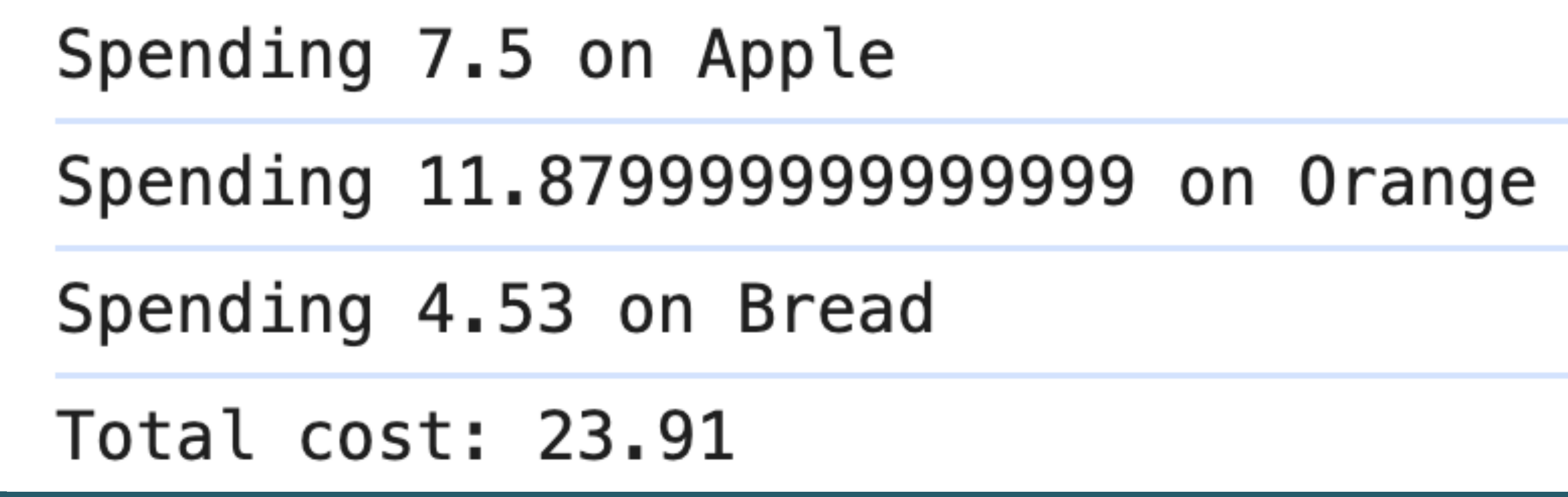
Putting it all together

```
let groceryItems = [
    {'name':'Apple', 'price': 1.25, 'numberToPurchase': 6},
    {'name':'Orange', 'price': 0.99, 'numberToPurchase': 12},
    {'name':'Bread', 'price': 4.53, 'numberToPurchase': 1}
];

let totalCost = 0;

for(let groceryItem of groceryItems) {
    let costOnItem = groceryItem.price * groceryItem.numberToPurchase;
    console.log('Spending ' + costOnItem + ' on ' + groceryItem.name);
    totalCost = totalCost + costOnItem;
}

console.log('Total cost: ' + totalCost);
```



```
Spending 7.5 on Apple
Spending 11.879999999999999 on Orange
Spending 4.53 on Bread
Total cost: 23.91
```

Putting it all together

```
let groceryItems = ['Apple', 'Orange', 
'Bread'];
let prices = [1.25, 0.99, 4.53];
let numberToPurchase = [6, 12, 1];

let totalCost = 0;

for(let i=0; i < groceryItems.length; i 
= i + 1) {
    let costOnItem = prices[i] * 
numberToPurchase[i];
    console.log('Spending ' + 
costOnItem + ' on ' + groceryItems[i]);
    totalCost = totalCost + costOnItem;
}

console.log('Total cost: ' + 
totalCost);
```

```
let groceryItems = [
    {'name':'Apple', 'price': 1.25, 
'numberToPurchase': 6},
    {'name':'Orange', 'price': 0.99, 
'numberToPurchase': 12},
    {'name':'Bread', 'price': 4.53, 
'numberToPurchase': 1}
];

let totalCost = 0;

for(let groceryItem of groceryItems) {
    let costOnItem = groceryItem.price * 
groceryItem.numberToPurchase;
    console.log('Spending ' + costOnItem + 
' on ' + groceryItem.name);
    totalCost = totalCost + costOnItem;
}

console.log('Total cost: ' + totalCost);
```

So, what if we want to repeat something in our code?

Repeating code

```
let groceryItems = [
    {'name':'Apple', 'price': 1.25, 'numberToPurchase': 6},
    {'name':'Orange', 'price': 0.99, 'numberToPurchase': 12},
    {'name':'Bread', 'price': 4.53, 'numberToPurchase': 1}
];
let totalCost = 0;
for(let groceryItem of groceryItems) {
    // ...
}
console.log('Total cost: ' + totalCost);

groceryItems.push({'name':'Cake', 'price':20.89, 'numberToPurchase':
2});
```

- Now what...?

Functions

- Functions allow you to write code which can be used many times
- You can use the same code with different *arguments* to produce different results

Functions

Function *name* Function *argument(s)*

```
function toCelsius(fahrenheit) {
    return (5/9) * (fahrenheit - 32);
  }
```

Value to *return*

```
let tempF = 77;
console.log(tempF + ' in Celsius is ' + toCelsius(tempF));
tempF = 32;
console.log(tempF + ' in Celsius is ' + toCelsius(tempF));
```

```
77 in Celsius is 25
32 in Celsius is 0
```

Functions

```
function calculateCost(groceryItem) {
    return groceryItem.price * groceryItem.numberToPurchase;
}

let groceryItems = [ //...
];

let totalCost = 0;

for(let groceryItem of groceryItems) {
    console.log('Spending ' + calculateCost(groceryItem) + ' on ' +
groceryItem.name);
    totalCost = totalCost + calculateCost(groceryItem);
}

console.log('Total cost: ' + totalCost);
```

Functions

Functions can have multiple arguments

```
function calculateCost(price, numberToPurchase) {
    return price * numberToPurchase;
}

let groceryItems = [ //...
];

let totalCost = 0;

for(let groceryItem of groceryItems) {
    console.log('Spending ' + calculateCost(groceryItem.price,
groceryItem.numberToPurchase) + ' on ' + groceryItem.name);
    totalCost = totalCost + calculateCost(groceryItem.price,
groceryItem.numberToPurchase);
}

console.log('Total cost: ' + totalCost);
```

Functions

```
function calculateCost(price, numberToPurchase) {
    return price * numberToPurchase;
}

function calculateTotal(items) {
    let totalCost = 0;
    for(let item of items) {
        console.log('Spending ' + calculateCost(item.price, item.numberToPurchase) + ' on ' + item.name);
        totalCost = totalCost + calculateCost(item.price, item.numberToPurchase);
    }
    return totalCost;
}

let groceryItems = [
    {'name':'Apple', 'price': 1.25, 'numberToPurchase': 6},
    {'name':'Orange', 'price': 0.99, 'numberToPurchase': 12},
    {'name':'Bread', 'price': 4.53, 'numberToPurchase': 1}
];

console.log('Total cost: ' + calculateTotal(groceryItems));

groceryItems.push({'name':'Cake', 'price':20.89, 'numberToPurchase': 2});

console.log('Total cost: ' + calculateTotal(groceryItems));
```

```
Spending 7.5 on Apple
Spending 11.879999999999999 on Orange
Spending 4.53 on Bread
Total cost: 23.91
Spending 7.5 on Apple
Spending 11.879999999999999 on Orange
Spending 4.53 on Bread
Spending 41.78 on Cake
Total cost: 65.69
```

Today’s goals

By the end of today, you should be able to...

- Use arrays in Javascript to list items
- Use objects in Javascript to organize content by keys and values
- Manipulate arrays and objects to add new values and change existing ones
- Loop over arrays and dictionaries to access data
- Create and call functions to separate out code

IN4MATX 285: Interactive Technology Studio

Programming: Data Structures and Functions in JavaScript

Additional slides

Useful array methods

- JavaScript arrays have stack functions
  - `.push()` and `.pop()` to add and remove the last item, respectively
- Arrays can be combined with `.concat()`
- `.sort()` will sort alphabetically/numerically by default
  - But can take in a comparator
  - For example, sort by the count attribute of an object:

```
array.sort(function(a, b) {
  return a.count - b.count;
});
```

https://medium.com/@DaphneWatson/10-useful-javascript-array-methods-8ffe22e7a959

Hoisting

- Variable and function declarations get *hoisted* to execute before the rest of the code
  - Assignment occurs later, where you specify it

```
bar();
var foo = 42;
function bar() {}
//=> is interpreted as
var foo;
function bar() {}
bar();
foo = 42;
```

https://stackoverflow.com/questions/7609276/javascript-function-order-why-does-it-matter

Functions

- In Javascript, all parameters are optional

```
function sayHello(name)
{
    return "Hello, "+name;
}

//expected; parameter is assigned a value
sayHello("In4MATX 133"); //"Hello, IN4MATX 133"

//parameter not assigned value (left undefined)
sayHello(); //"Hello, undefined"

//extra parameters (values) are not assigned
//to variables, so are ignored
sayHello("IN4MATX","133"); //"Hello, IN4MATX"
```