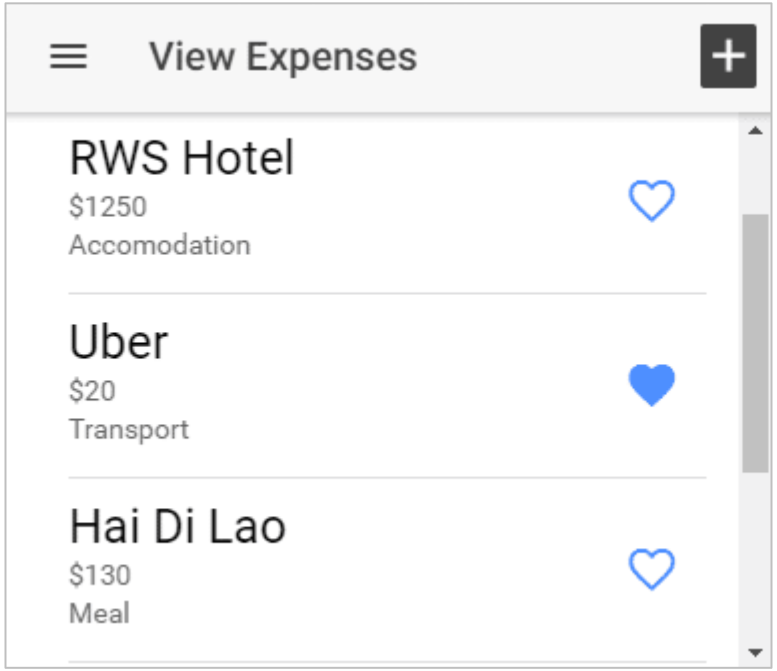
3 Lists

List & Item

- **Lists** contain **items**
- An **item** can contain **text**, **icons**, **images**, and anything else.

```
<ion-list>
  <ion-item>
    Item 1
  </ion-item>
  <ion-item>
    Item 2
  </ion-item>
</ion-list>
```


View Expenses

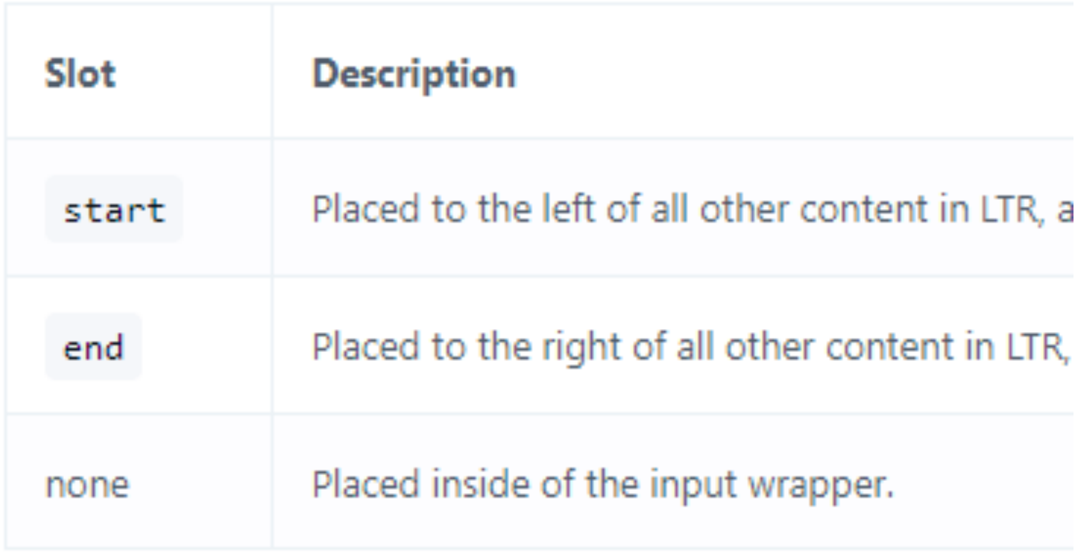
RWS Hotel
$1250
Accomodation

Uber
$20
Transport

Hai Di Lao
$130
Meal

Slot

Item uses named slots order to position content



Slot	Description
`start`	Placed to the left of all other content in LTR, a
`end`	Placed to the right of all other content in LTR,
none	Placed inside of the input wrapper.

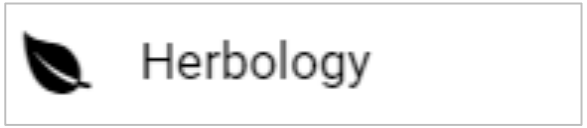
#1 Icon

Use name to specify which icon is used.

https://ionicons.com/

Icons can be used **on their own**, or **inside other Ionic components**.

```
<ion-item>
    Herbology
  <ion-icon name="leaf"></ion-icon>
</ion-item>
```

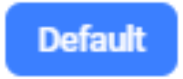


#2 Button

Button may display text, icons, or both.

```
<ion-button> Default </ion-button>
```

```
<ion-button>
  <ion-icon slot="icon-only" name="star"></ion-icon>
</ion-button>
```

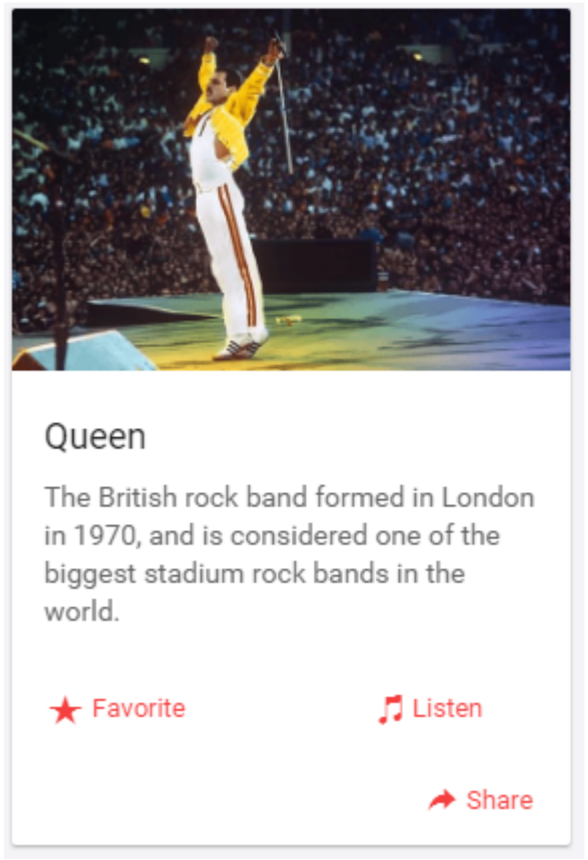


#3 Image

```
<img src="…">
```

```
<ion-img src="…"></ion-img>
```

<ion-img> is a tag that will **lazily** load an image when ever the tag is in the viewport. This is extremely useful when generating a large list as images are only loaded when they're visible.


Queen

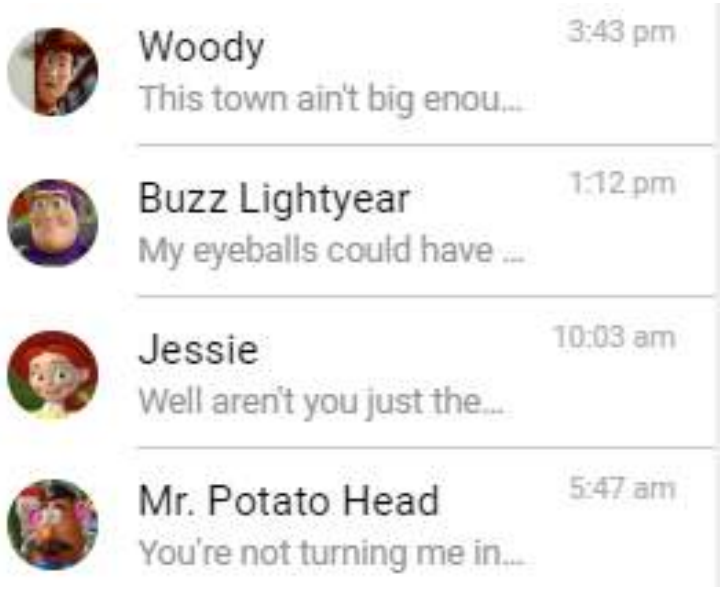
The British rock band formed in London in 1970, and is considered one of the biggest stadium rock bands in the world.

Image

- The ion-img component is similar to the standard img element, but it also adds features in order to provide improved performance.
- Features include:
    - Only loading images which are visible.
    - Using web workers for HTTP requests.
    - Preventing jank while scrolling.
    - In-memory caching.
- A good rule is, if there are **only a few images** to be rendered on a page, then the standard img is probably best.
- However, if a page has the potential for **hundreds or even thousands of images** within a scrollable area, then ion-img would be better suited for the job.

Avatar

```
<ion-item>
  <ion-avatar slot="start">
    <img src="img/avatar.png">
   </ion-avatar>
  <h2>Woody</h2>
  <p>This town aren't big enough.</p>
</ion-item>
```



Woody 3:43 pm
This town ain't big enou...

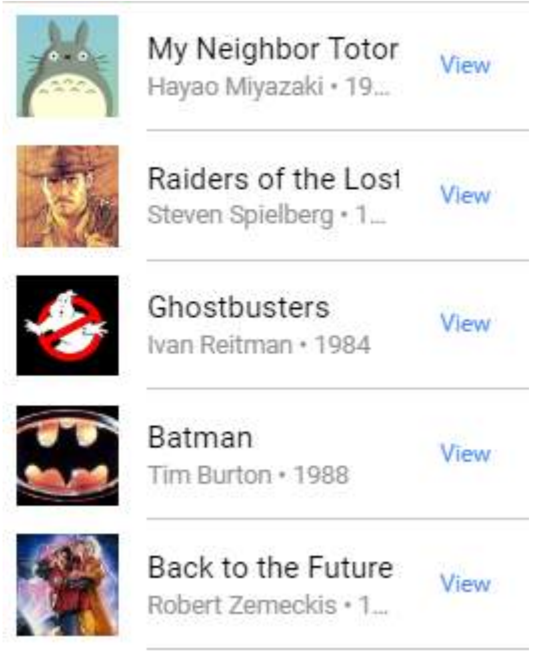
Buzz Lightyear 1:12 pm
My eyeballs could have ...

Jessie 10:03 am
Well aren't you just the...

Mr. Potato Head 5:47 am
You're not turning me in...

Thumbnail

```
<ion-item>
    <ion-thumbnail slot="start">
        <img src="img/thumbnail-totoro.png">
    </ion-thumbnail>
    <h2>My Neighbor Totoro</h2>
    <p>Hayao Miyazaki • 1988</p>
    <button ion-button clear item-end>View</button>
</ion-item>
```



My Neighbor Totor
Hayao Miyazaki • 19...
View

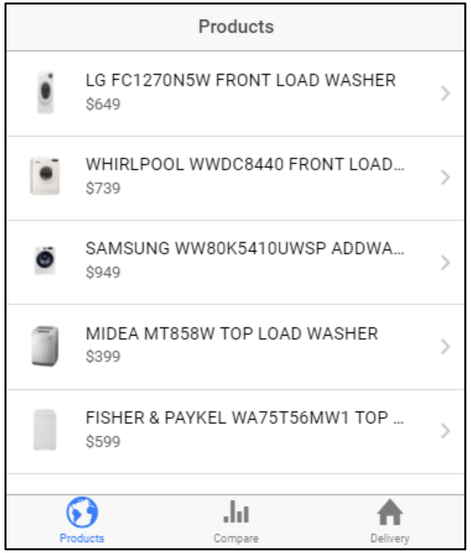
Raiders of the Lost
Steven Spielberg • 1...
View

Ghostbusters
Ivan Reitman • 1984
View

Batman
Tim Burton • 1988
View

Back to the Future
Robert Zemeckis • 1...
View

<ion-list> & <ion-item>


Products

LG FC1270N5W FRONT LOAD WASHER
$649

WHIRLPOOL WWDC8440 FRONT LOAD...
$739

SAMSUNG WW80K5410UWSP ADDWA...
$949

MIDEA MT858W TOP LOAD WASHER
$399

FISHER & PAYKEL WA75T56MW1 TOP ...
$599

```
<______________>
    <______________ color="none" >
      <_____________________ slot="start">
        <img src="assets/lg_washine-machine.jpg" />
      </…>
      <ion-label>
        <h2> LG Washing Machine </h2>
        <p> $500 </p>
      </ion-label>
    </…>
</…>
```

HTML

- <ion-list>
- <ion-item>

- <ion-icon>
- <ion-img>
- <ion-button>
- <ion-label>

- <ion-avatar>
- <ion-thumbnail>

TypeScript Class

Basic Types

TS Types
- boolean
- number
- string
- enum
- any

Array
[]

Variable Declarations

`let` is similar to `var` without all the quirks of `var` declarations in JavaScript, i.e. better than `var`.

```
let isDone: boolean = false;
let decimal: number = 6;
let color: string = "blue";

let list: number[] = [1, 2, 3];
let list: Array<number> = [1, 2, 3];

let notSure: any = 4;
notSure = "maybe a string instead";
```

Var, let, const

	var	let	const
Scope	Global / Function	Block	Block
Reassign	Y	Y	N
When to use	Global variable	Temporary variable used in a function or loop	Constant that cannot be changed, e.g PI

Class

TypeScript adds object-oriented class approach to JavaScript

```
export class HomePage {
  products: string[];
}
```

Class Members

```
export class SubmitExpensePage {
  categories: string[];

  constructor(public navCtrl: NavController) {
    …
  }

  onSubmit(form: NgForm) {
    …
  }

}
```

- This class `SubmitExpensePage` has 3 **members**
  - A property called `categories`
  - A constructor
  - A method called `onSubmit`

#1 Property

- All members are public by default.
- In this course for simplicity, all class members are declared public so we can directly access them.
- You can choose to write your own accessors (getters/setters) which is not covered here.

```
export class User {
  username: string;
  password: string;
}
```

#2 Constructor

```
export class User {

  username: string;
  password: string;

  constructor(username: string,  password: string) {
    this.username = username;
    this.password = password;
  }

}
```

- Constructor is called by the **new** keyword
- **this** is used to refer to class members

#3 Method

```
export class SubmitExpensePage {
  …

  onSubmit(form: NgForm) {
    …
  }

}
```

- Method name – `onSubmit`
- Method parameter: `form`

Parameter Property

```
export class Expense {
   constructor(
      public date: string,
      public amount: number,
      public category: string,
) { … }
}
```

How many **properties** does the class Expense have?

- Parameter properties are declared by prefixing a **constructor** parameter with an accessibility modifier or readonly, or both.
- Using **`public`** for a parameter property declares and initializes a **`public`** member; likewise, the same is done for **`private`**, **`protected`**, and **`readonly`**.

Optional Parameter

```
export class Expense {
  constructor(
    public date: string,
    public amount: number,
    public category: string,
    public merchant: string,
    public notes?: string) {
    …
  }
}
```

Use ? For optional parameters in **constructor** and **methods**

new

```
export class Expense {

  status: string;
  user: string;

  constructor(
    public date: string,
    public amount: number,
    public  category: string,
    public merchant: string,
    public notes?: string,
    public favIcon?: string) {
    if (!this.favIcon)
      this.favIcon = '';
    if (!this.notes)
      this.notes = '';
    this.status = "pending";
  }

}
```

New Object

- Use the `new` keyword to create an expense object

```
this.expense = new Expense('1/1/2021', 15,
'Transport', 'Grab');


this.expense = new Expense('1/1/2021', 15,
'Transport', 'Grab', 'Travel to meeting', 'heart');
```

this

this is used to refer to class members

```
export class SubmitExpensePage {
  …

  onSubmit(form: NgForm) {
    alert("Date: " + this.date + "Amount" + this.amount);
    …
  }

}
```

Export & Import

Export

- Any declaration (class, interface, function, variable, type alias) can be exported by adding the `export` keyword.

```
export class SubmitExpensePage {
  …

}
```

Import

- Importing an exported declaration is done through using the `import` keyword.

```
import { SubmitExpensePage } from
'../pages/submit-expense/submit-
expense';
```

*ngFor
*ngIf

ng stands for Angular

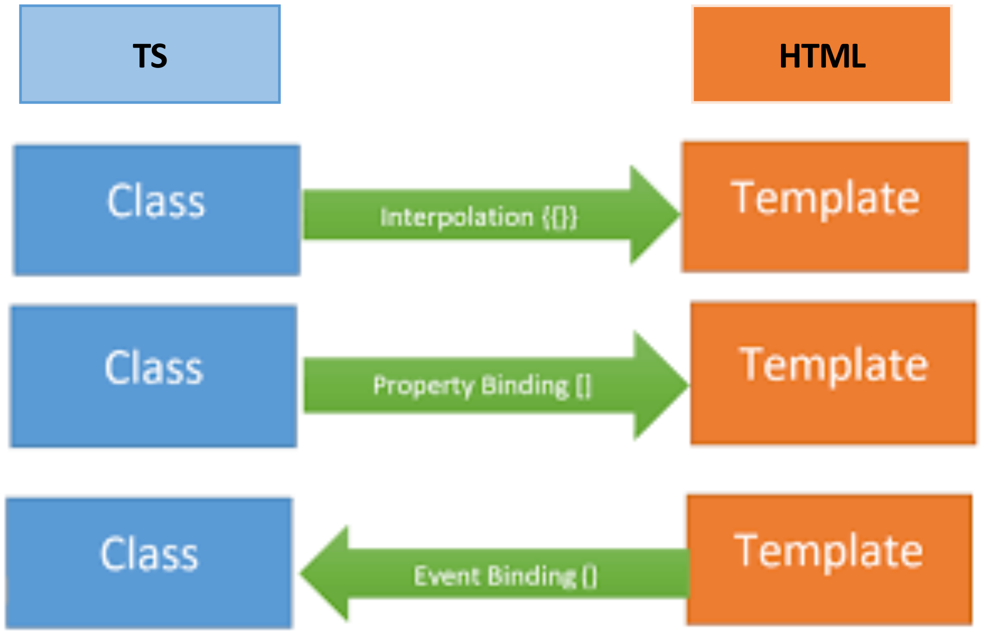
Data Binding

HTML talking to TS

{{ }}

[ ]

( )



*ngFor

HTML
*ngFor = “let … of …”

Expense

merchant
amount
category
date

TS

```
export class ViewExpensesPage {
  expenses: Expense[];
  …
}
```

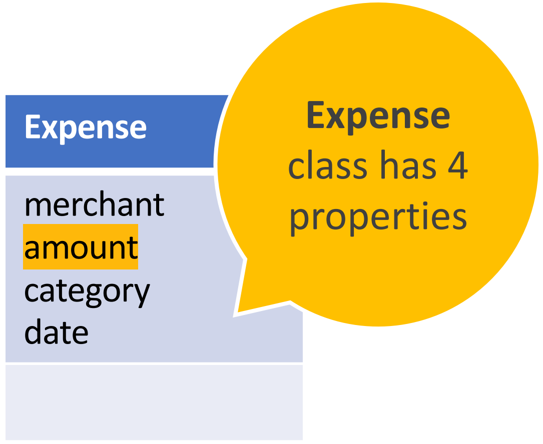
HTML

```
<ion-list>
  <ion-item *ngFor = "let item of expenses">
    …
  </ion-item>
</ion-list>
```

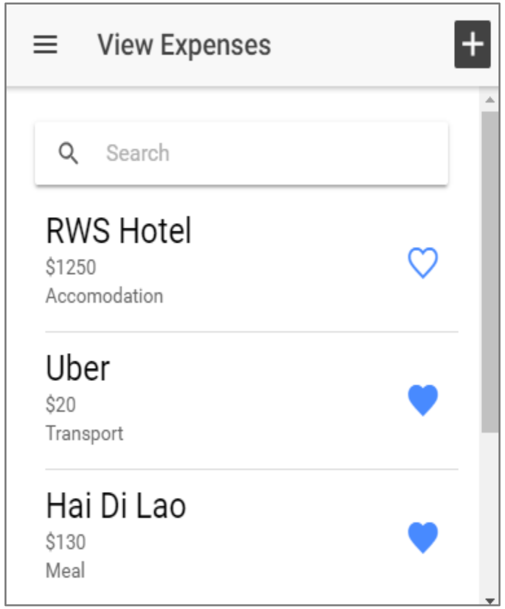
{{ }} Interpolation

- Show a **property** by binding the property name through interpolation `{{...}}`.
- With interpolation, you put the property name in the view template, enclosed in double curly braces: `{{item.amount}}`.

```
<ion-list>
  <ion-item *ngFor = "let item of expenses">
    <p> ${{item.amount}}  </p>
  </ion-item>
</ion-list>
```



*ngFor


View Expenses

Search

RWS Hotel
$1250
Accomodation

Uber
$20
Transport

Hai Di Lao
$130
Meal

```
<ion-list>
  <ion-item *ngFor = "let item of expenses">
    <h1> {{__________.merchant}} </h1>
    <p> ${{__________.amount}} </p>
    <p> {{__________.category}} </p>
  </ion-item>
</ion-list>
```

Expense
merchant amount category Date

[src] Property Binding

Interpolation {{ }} is commonly used for text only.

To bind to **HTML attribute**, use [ ] property binding.

```
<img src="unicorn.png">
<img [src]="expense.image">
```

- This property binding passes the value of `"expense.image"` to the `"src"` HTML attribute.

```
<input type="date" value="1/1/2022">
<input type="date" [value]="expense.date">
```

- This property binding passes the value of `"expense.date"` to the `"value"` HTML attribute.

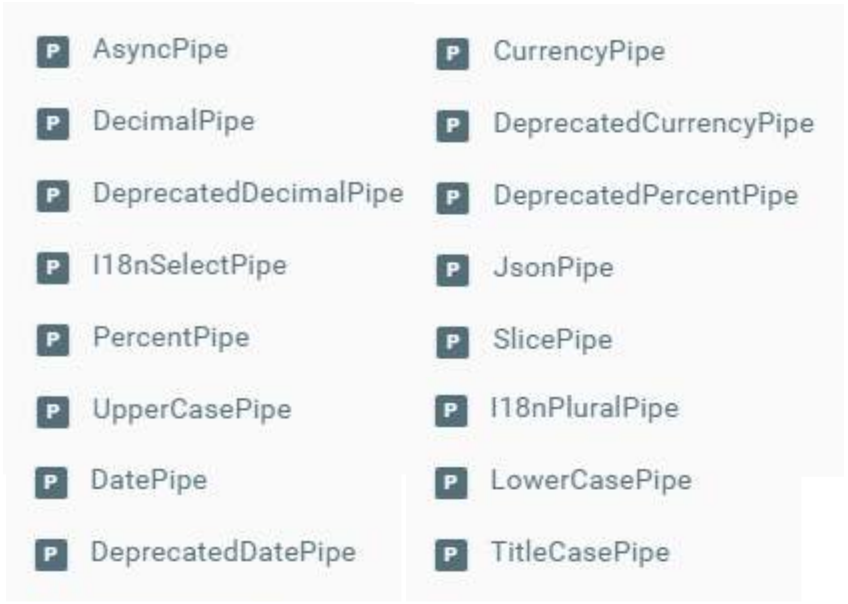
| Pipe

- A **pipe** takes in data as input and transforms it to a desired output.

```
{{expense.date | date:"dd MMMM yy, h.m a"}}
{{expense.amount | currency }}
```

- Inside the interpolation expression, you flow the expense date value through the pipe operator (|) to the **Date** pipe function on the right.

- Angular comes with a stock of pipes such as **DatePipe**, **UpperCasePipe**, **LowerCasePipe**, **CurrencyPipe**, and **PercentPipe**. They are all available for use in any template.



P AsyncPipe	P CurrencyPipe
P DecimalPipe	P DeprecatedCurrencyPipe
P DeprecatedDecimalPipe	P DeprecatedPercentPipe
P I18nSelectPipe	P JsonPipe
P PercentPipe	P SlicePipe
P UpperCasePipe	P I18nPluralPipe
P DatePipe	P LowerCasePipe
P DeprecatedDatePipe	P TitleCasePipe

*ngIf

The *ngIf on the HTML element shows only if true.

```
<div id="container" *ngIf="products.length===0">
   <strong>Add a new product</strong>
</div>
```