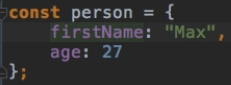
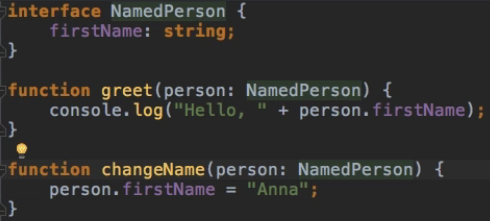
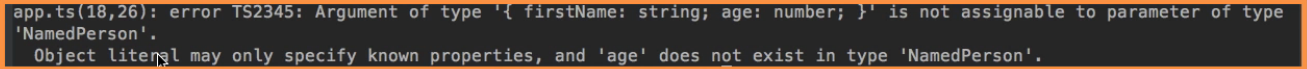
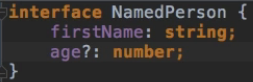
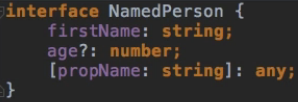
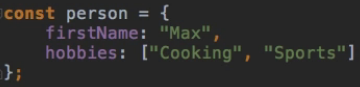
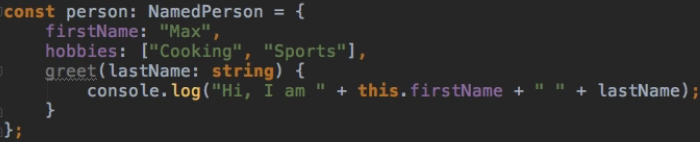
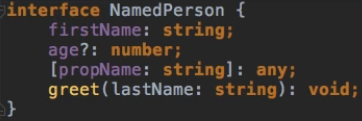
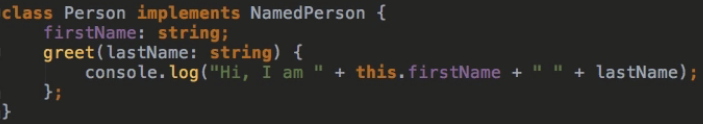
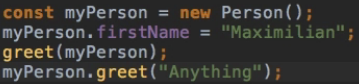
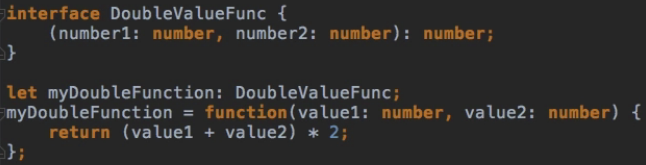
**Introduction**  
\* So far we learned about a lot of the core features.  
\* Now it’s time to also come to some features which are brand new, brand new introduced by TypeScript, something we didn’t really know before from the JavaScript world because modules, classes and so on, these are all things we already knew from JavaScript even though we might have had some different approaches of getting there, but it’s not brand new.  
\* This module looks at Interfaces, which really is a concept you didn’t know before in JavaScript.

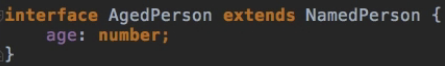
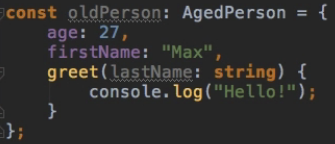
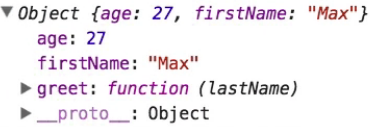
**The Basics about Interfaces**  
  
  
\* Now let’s say we change it to firstName:  
  
  
\* **For situations like these when we have whatever code which needs to know that what it uses has a certain property or method or whatever, then we might need an interface**.  
\* Interface basicaly is a contract signed by an object which says: I guarantee you that I have a certain property/function/whatever.  
  
=> **This means that it needs to at least have a `name` property which is string**.  
\* That’s basically what an interface is.  
\* **We define a type where we want to make sure that a certain field is available**.  
\* There’s a better way to create it:  
**interface**  
=> In there you define how this contract should look - what your requirements are.  
  
\* **Interfaces are a way to guarantee your code, that certain properties or methods or whatever are available**.

**Interfaces and Properties**  
  
  
\* **If you pass an Object Literal directly, it must have only what’s defined in the interface**.  
\* **If you assign the Object Literal to a constant first, then it’s not checked that strictly**.   
**? => optional argument in interface.**  
  
\* **What if you don’t know the names of the properties in advance?**  
**[name: type]**  
  
\* **It doesn’t have anything to do with an array, it’s just the TypeScript notation for setting up a flexible key name**.  


**Interfaces and Methods**  
  


**Using Interfaces with Classes**  
**implements**  
\* **We need to implement all the things we require in the interface in our class too - except the optional**.  
  
  
  
\* **If we implemented something that’s not in the interface, that’s okay**.  
\* Just as we were able to use extra properties when creating the object in a constant.  
\* Interface is a contract which can be signed/used as a type and which then makes sure all the conditions set up in the interface have to be fulfilled by whatever implements/uses this interface, whatever is of this type.

**Interfaces and Function Types**  
\* So we covered interfaces which define properties and methods and how to use them on objects and classes, or here as a type for our argument which gets passed to the greet() function.  
=> These are the most common use cases.  
\* But you can also create interfaces for Function Types.  
  
=> Whatever uses this interface, must be a function of this type.  
  
  
\* **That’s how you can use interfaces to set the type of functions**.

**Interface Inheritance**  
  
=> Now the age is required.  
  
  


**What Happens once Interfaces get Compiled**  
\* **Interfaces don’t get compiled at all**.  
\* **They’re just there to check your code during compilation to give you compilation errors**.

**Module Summary**  
\* You learned what interfaces are.  
\* How they can help you write better code, knowing what to expect in certain classes or functions and making sure that your dynamic code is able to work together just fine and always know which functions or types to expect and not to expect.