**TS**

**ts\_1. Что такое TypeScript? Для чего он нужен и какие проблемы решает?**

«TypeScript — это надмножество JavaScript» или «TypeScript — это типизированное расширение JavaScript»

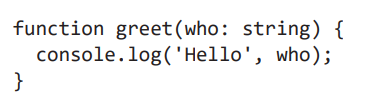
В TS добавляется система типов, и запрещаются некоторые сомнительные конструкции JavaScript, например вызов функций с неправильным количеством аргументов.

TypeScript - не выполняется в интерпретаторе (как Ruby или Python) и не компилируется в низкоуровневый язык (как Java или C). Программа транспилируется в другой высокоуровневый язык — JavaScript. Поэтому выполняется именно JavaScript, а не TypeScript.

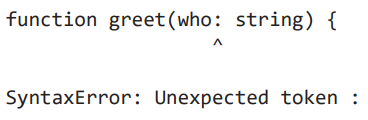
На высоком уровне tsc (TS-компилятор) делает две вещи: преобразует новейшую версию TypeScript/JavaScript в более старую версию JavaScript, которая работает во всех браузерах (транспиляция); проверяет код на наличие ошибок типов.

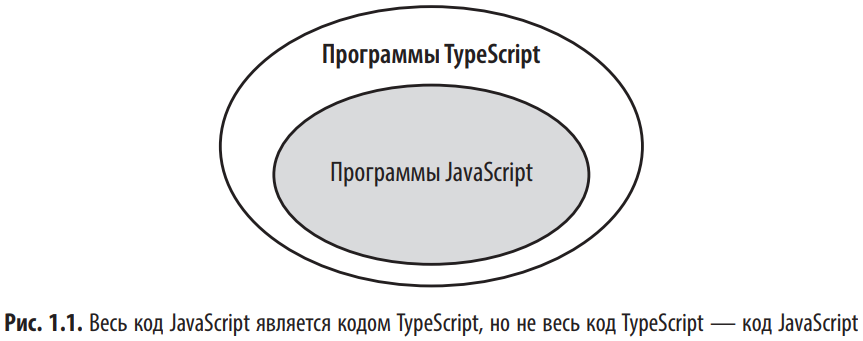
Все программы JS являются программами TS, но обратное справедливо не всегда: есть программы TS, которые не являются программами JS, из-за того, что в TypeScript добавлен дополнительный синтаксис определения типов.

Например, рабочая TS прога, не запустится через среду выполнения JavaScript:



Получится ошибка, т.к. : string является аннотацией типа, относящейся к TS, которая выходит за рамки базового JavaScript:





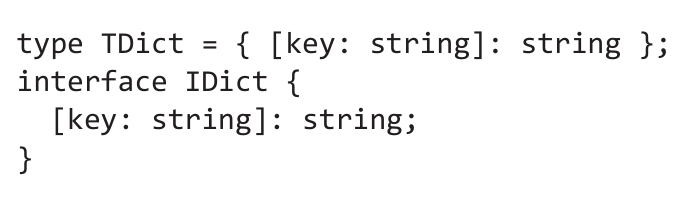
«Типы — это множества значений и то, что с ними можно сделать

**Что такое Type Aliases? Зачем нужны? Чем отличаются от интерфейсов?**

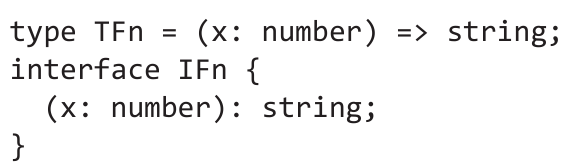
Что общего?

- проверка лишних свойств в литералах объектов

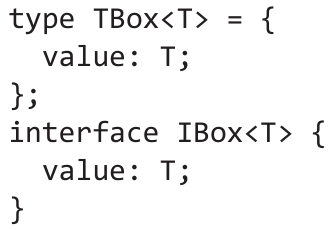
- может использоваться сигнатура индекса



- можно определять типы функций



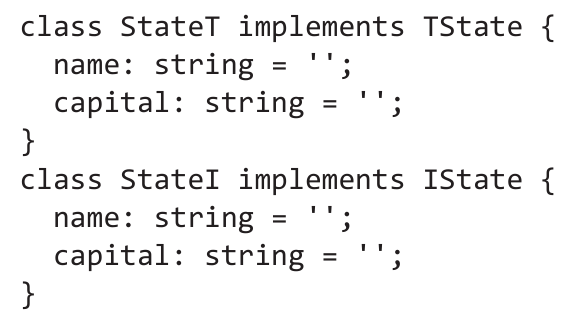
- могут быть обобщенными



- могут расширять друг друга, с некоторыми оговорками

interface может расширять только объектные типы, которые могли бы быть определены с interface. Например, не допускается расширение типов объединений. Если вам нужно именно это, используйте type и &.

- могут реализовывать классы

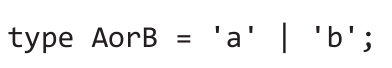


- могут быть рекурсивными

Отличия?

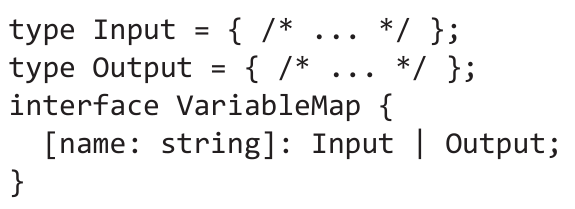
В целом type предоставляет больше возможностей, чем interface. Он может выступать в качестве объедине ния, а также пользоваться более продвинутыми возможностями вроде отобра женных типов (правило 15) или условных типов (правило 52).

- в типах могут содержаться объединения, в интерфейсах нет



- интерфейс не может расширять типы объединений. (Некоторые типы интерфейс может расширить).

Если у вас есть раздельные типы для переменных Input и Output и отображения из имени в переменную:

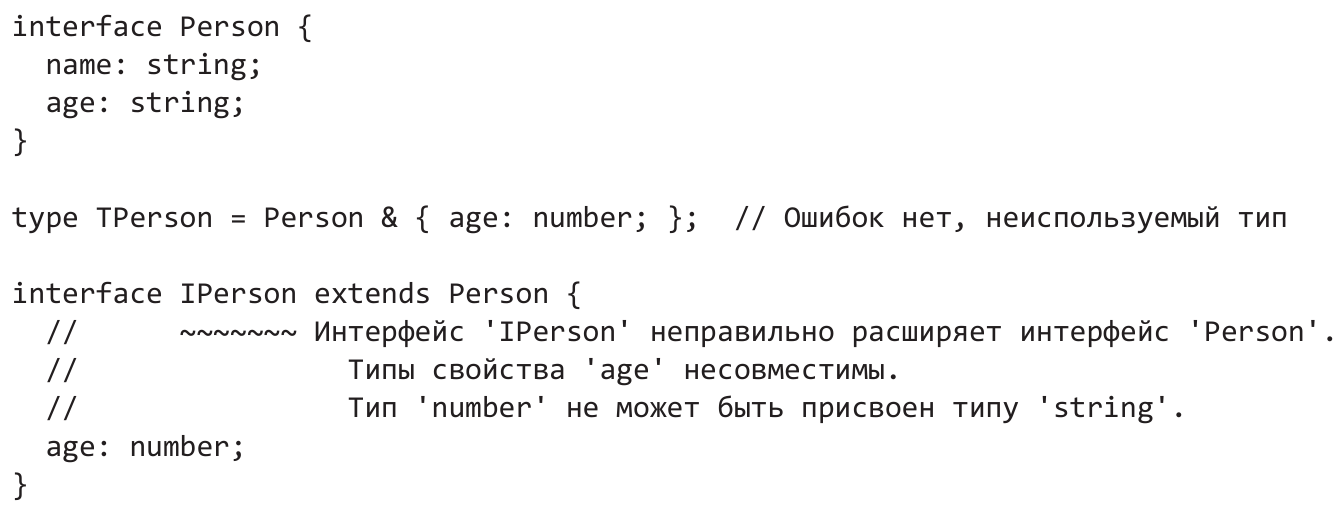


может понадобиться тип, который связывает имя с переменной. Например:



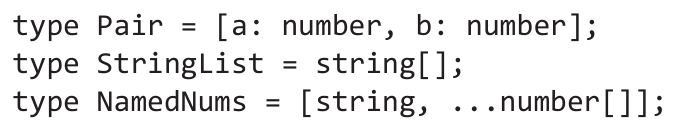
Этот тип не может быть выражен через interface.

- interface и extends обеспечивают чуть более высокий уровень проверки ошибок, чем type и &:

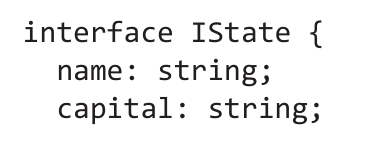


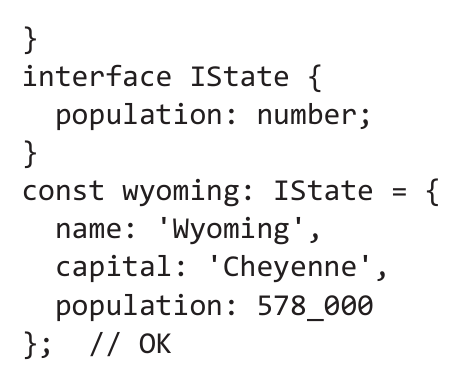
Изменение типа свойства в подтипе допустимо при условии, что он совместим с базовым типом (см. правило 7). Желательно иметь больше проверок безопас ности, так что это веская причина для использования extends c interface.

- Псевдонимы типов позволяют естественно выражать типы кортежей и массивов(а интерфейсы нет чтоли?):



- interface может быть дополнен, а type нет – этот синтаксис называется «слиянием объявлений» (declaration merging). Это может происходить в пользовательском коде, но только если два интерфейса определяются в одном модуле (то есть в одном файле .ts). Тем самым предотвращаются случайные конфликты с глобальными интерфейсами, имеющими такие общие имена, как Location и FormData.





Как и для каких целей используются типы данных, приведенные ниже?

* Any

Используйте any, если действительно хотите отключить проверку типа,

* Unknown

Используйте unknown, если вам нужно соблюдать осторожность.