تعلم برمجة
TypeScript

ابو حبيب الحسينى

فهرس الكتاب

مقدمة قصيره عن لغة تايب سكريبت

هى لغة تم انشائها من شركة مايكروسوفت تعمل بمحرك جافاسكرب يعنى تترجم للغة جافاسكربت فى النهاية وهى شبيها بالجافاسكربت الى حد كبير غير انها تضيف امكانات جديدة للغة وقامت شركة مايكرو سوفت بانشاء العديد من التطبيقات التى تعمل على جميع المنصات باستخدام لغة تايب سكربت اشهرها تطبيق فيجوال ستوديو كود الذى سنعمل علية فهو مصنوع بلغة تايب سكربت وذو اداء ممتاز وشهرة عالميه فانت تستطيع ان تنشا برامج تعمل على جميع انظمة التشغيل سواء موبيل او كمبيوتر باستخدام لغة التايب سكربت ودمج تقنيات الويب العادية من اتش تى ام ال وسى اس اس فى مشروعك وتستطيع دمج مكتبات نودى جى اس فى مشروعك لتتوسع فى العمل حيث ان النود جى اس هى من افضل التقنيات البرمجية فى تاريخ البشرية التى لا حصر للمميزاتها وهذا فى رائيى الشخى وسننشا كتاب خاص لشرح مميزات وعمل النود جى اس تفصيليا ان شاء الله تعالى

يعنى باختصار

تايب سكربت هو جافاسكربت مع إضافة بناء جملة للأنواع. المختلة من البيانات والاضافات والاجرائات التى تستطيع استخدمها فى انتاج تطبيقات لجميع المنصات

تايب سكربت عبارة عن مجموعة شاملة تنسيقية من **جافاسكربت** تضيف كتابة ثابتة . **للكود الذى يسهل العمل على المطور كافضل من جافاسكربت**

. هذا يعني في الأساس أن **تايب سكربت** يضيف بناء الجملة أعلى **جافاسكربت**، مما يسمح للمطورين بإضافة أنواع

يعني أنه يشترك في نفسالعديد من البنية المحجوزه للجافا **"Syntactic Superset"** كون **تايب سكربت** عبارة عن **جافاسكربت**، ولكنه يضيف شيئًا إليها.

واعلم ايضا

جافا سكريبت هي لغة مكتوبة بشكل فضفاض. قد يكون من الصعب فهم أنواع البيانات التي يتم تمريرها في جافاسكربت. وهذا ثابت فالكثير من الناس يشتكون من صعوبة جافاسكربت

في جافا سكريبت، لا تحتوي معلمات ومتغيرات الدالة على أي معلومات! لذلك يحتاج المطورون إلى إلقاء نظرة على الوثائق، أو التخمين بناءً على التنفيذ.

يسمح تايب سكربت بتحديد أنواع البيانات التي يتم تمريرها داخل التعليمات البرمجية، ولديه القدرة على الإبلاغ عن الأخطاء عندما لا تتطابق الأنواع.

على سبيل المثال، سيبلغ تايب سكربت عن خطأ عند تمرير نص إلى دالة تتوقع رقمًا. جافا سكريبت لن.

يستخدم تايب سكربت التحقق من نوع وقت الترجمة. مما يعني أنه يتحقق من تطابق الأنواع المحددة قبل تشغيل الكود، وليس أثناء تشغيل الكود وهذا على عكس جافا سكربت.

كيف يمكنني استخدام تايب سكريبت؟

إحدى الطرق الشائعة لاستخدام **تايب سكربت** هي استخدام مترجم **تايب سكربت** الرسمي، الذي ينقل كود **تايب سكربت** إلى **جافاسكربت**.

يوضح القسم التالي كيفية الحصول على إعداد المترجم لمشروع محلي.

تحتوي بعض برامج تحرير التعليمات البرمجية الشائعة، مثل **فيجوال ستوديو كود**، على دعم **تايب سكربت** مدمج ويمكنها إظهار الأخطاء أثناء كتابة التعليمات البرمجية!

البدء باستخدام تايب سكربت

كيف يعمل مترجم تايب سكريبت

يتم تحويل **تايب سكربت** إلى **جافاسكربت** باستخدام برنامج التحويل البرمجي.

تحويل **تايب سكربت** إلى **جافاسكربت** يعني أنه يعمل في أي مكان يتم فيه تشغيل **جافاسكربت! على اى منصة**

من عيوب تايب سكربت

في بعض الأحيان، لا تتم صيانة المشاريع،من كثرة حزم ان بى ام حيث صنفة حزم ان بى ام انها اكبر مستودع حزم برمجية فى العالم وهذا يرجع الى شهرة جافاسكربت وعدد مستخدميها الذى لا حصر له وفي أحيان أخرى ل تكون الحزم متوافقة مع تايب اسكربت او يكونون غير مهتمين باستخدام **تايب سكربت حيث ان شركة مايكروسفت هى التى تتولى تطويره ولا تعطية الاهتمام الاكبر لانه مجانى وبعض المطورين المتطوعين الذى يهتمون بهذا الاطار** ولكن هذة العيوب قد تتغير فى لحظة . فقد يكثر عدد مستحدمى التايب سكربت ولا نعرف ماذا يحدث غدا

استقلال تايب اسكربت

لن يكون استخدام حزم ان بى ام غير المكتوبة مع **تايب سكربت** آمنًا للكتابة بسبب نقص الأنواع.

لمساعدة مطوري **تايب سكربت** على استخدام مثل هذه الحزم، يوجد مشروع يديره المجتمع يسمى **Definitely Typed** .

Definitely Typed هو مشروع يوفر مستودعًا مركزيًا لتعريفات **تايب سكربت** لحزم ان بى ام التي لا تحتوي على أنواع.

مثال

```
npm install --save-dev @types/jquery
```

لا توجد عادةً حاجة إلى خطوات أخرى لاستخدام الأنواع بعد تثبيت حزمة الإقرارات، وسيقوم **تايب سكربت** تلقائيًا باختيار الأنواع عند استخدام الحزمة نفسها.

غالبًا ما يقترح المحررون مثل **فيجوال ستوديو كود** تثبيت حزم مثل هذه عندما تكون الأنواع مفقودة.

سنستخدم برنامج فيجوال ستوديو كود

قم بتنصيب اضافة التكملة التلقائية اوتوكومليت تايب سكربت فى المحرر فيجوال ستوديو كى تساعدك فى كتابة الاكواد

يمكنك تعديل كود **تايب سكربت** وعرض النتيجة في متصفحك.**باستخدام اى اضافة سيرفر للفيجوال ستوديو كود**

تثبيت المترجم

يحتوي **تايب سكربت** على مترجم رسمي يمكن تثبيته من خلال **npm**.

قم بتشغيل الأمر التالي لتثبيت المترجم ، **npm** ضمن مشروع:

```
npm install typescript --save-dev
```

والتي يجب أن تعطيك مخرجات مشابهة فى:

```
added 1 package, and audited 2 packages in 2s
found 0 vulnerabilities
```

تم تثبيت المترجم في `node_modules`الدليل ويمكن تشغيله باستخدام: `npx tsc`.

```
npx tsc
```

والتي يجب أن تعطيك مخرجات مشابهة فى:

```
Version 4.5.5
tsc: The  typescript Compiler - Version 4.5.5
```

تليها قائمة بجميع الأوامر المشتركة.

كيف يعمل المترجم

افتراضيًا، سيقوم مترجم **تايب سكربت** بطباعة رسالة مساعدة عند تشغيله في مشروع فارغ.

يمكن تكوين المترجم باستخدام ملف`tsconfig.json`.

يمكنك إنشاء **تايب سكربت** `tsconfig.json`باستخدام الإعدادات الموصى بها باستخدام:

```
npx tsc --init
```

والتي يجب أن تعطيك مخرجات مشابهة فى:

```
Created a new tsconfig.json with:
TS
  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true

You can learn more at https://aka.ms/tsconfig.json
```

فيما يلي مثال على المزيد من الأشياء التي يمكنك إضافتها إلى الملف `tsconfig.json`:

```
{
  "include": ["src"],
  "compilerOptions": {
    "outDir": "./build"
  }
}
```

يمكنك فتح الملف في محرر لإضافة هذه الخيارات. سيؤدي هذا إلى تكوين برنامج التحويل البرمجي فى **تايب سكربت** لنقل الدليل `build/` دليل مشروعك إلى ملفات **جافاسكربت** في `src/` ملفات **تايب سكربت** الموجودة في.

هذه إحدى الطرق للبدء بسرعة في استخدام **تايب سكربت**، وهناك العديد من الخيارات الأخرى المتاحة مثل قالب إنشاء تطبيق تفاعلي ومشروع بداية نود جى اس ومكون إضافي لحزمة الويب .

انواع البيانات البسيطة فى تايب سكربت

يدعم **تايب سكربت** بعض الأنواع البسيطة (الأولية) التي قد تعرفها.

هناك ثلاثة أساسيات رئيسية في **جافاسكربت** و**تايب سكربت**.

- `boolean`- قيم صحيحة أو خاطئة
- `number`- الأعداد الصحيحة وقيم الفاصلة العائمة
- `string`- قيم نصية مثل "**تايب سكربت Rocks**"

انشاء المتعير على نوعين

عند إنشاء متغير، هناك طريقتان رئيسيتان لتعيين **تايب سكربت** للنوع:

- صريح
- ضمني

`firstName`هو من النوع`string` في كلا المثالين أدناه

النوع الصريح

صريح – كتابة النوع:

```
let firstName: string = "Dylan";
```

كما فى لغة الاكشن سكربت

إن تعيين النوع الصريح أسهل في القراءة وأكثر قصدًا.

النوع الضمني

ضمني – سوف "يخمن" **تايب سكربت** النوع، بناءً على القيمة المخصصة:

```
let firstName = "Dylan";
```

ملاحظة: إن قيام **تايب سكربت** بتخمين نوع القيمة يسمى الاستدلال .

يفرض التعيين الضمني على **تايب سكربت** استنتاج القيمة.

تكون مهمة الكتابة الضمنية أقصر وأسرع في الكتابة، وغالبًا ما تستخدم عند التطوير والاختبار.

خطأ في نوع التعيين

سوف يلقي **تايب سكربت** خطأً إذا كانت أنواع البيانات غير متطابقة.

مثال

```
let firstName: string = "Dylan"; // type string
firstName = 33; // attempts to re-assign the value to a different
type
```

لكن كلاهما سيلقي خطأ، أقل وضوحًا كـ string كان تعيين النوع الضمني firstName:

مثال

```
let firstName = "Dylan"; // inferred to type string
firstName = 33; // attempts to re-assign the value to a different
type
```

لن تقوم **جافاسكربت** بإصدار خطأ للأنواع غير المتطابقة.

احذر خطا غير قادر على الاستنتاج

الذي يؤدي إلى any قد لا يستنتج **تايب سكربت** دائمًا نوع المتغير بشكل صحيح. في مثل هذه الحالات، سيتم تعيين النوع تعطيل التحقق من النوع.

مثال

```
// Implicit any as JSON.parse doesn't know what type of data it
returns so it can be "any" thing...
const json = JSON.parse("55");
// Most expect json to be an object, but it can be a string or a
number like this example
console.log(typeof json);
```

noImplicitAny كخيار في مشروع **تايب** يمكن تعطيل هذا السلوك عن طريق تمكينه
سكربت tsconfig.json. هذا هو ملف تكوين **JSON تايب سكربت** لتخصيص كيفية تصرف بعض أنواع.

Boolean. ملاحظة: قد ترى أنواعًا بدائية مكتوبة بأحرف كبيرة مثل

boolean !== Boolean
في هذا الكتاب ، عليك فقط معرفة كيفية استخدام القيم ذات الأحرف الصغيرة، والقيم الكبيرة مخصصة لظروف محددة جدًا.

أنواع خاصة من تايب سكربت

يحتوي **تايب سكربت** على أنواع خاصة قد لا تشير إلى أي نوع محدد من البيانات.

النوع: أي

any هو نوع يعطل التحقق من النوع ويسمح باستخدام جميع الأنواع بشكل فعال.

المثال أدناه لا يستخدم any وسوف يلقي خطأ:

مثال بدون any

```
let u = true;
u = "string"; // Error: Type 'string' is not assignable to type
'boolean'.
Math.round(u); // Error: Argument of type 'boolean' is not
assignable to parameter of type 'number'.
```

إلى تعطيل التحقق من النوع any إلى النوع الخاص any يؤدي الإعداد:

مثال مع any

```
let v: any = true;
v = "string"; // no error as it can be "any" type
Math.round(v); // no error as it can be "any" type
```

يمكن أن تكون طريقة مفيدة لتجاوز الأخطاء نظرًا لأنها تعطل التحقق من النوع، لكن **تايب سكربت** لن يكون قادرًا على any توفير الأمان للكتابة، ولن تعمل الأدوات التي تعتمد على بيانات النوع، مثل الإكمال التلقائي. تذكر أنه يجب تجنبه بأي ثمن...

استخدام النوع: غير معروف

unknown هو بديل مشابه ولكنه أكثر أمانًا فى any.

استخدام الأنواع، كما هو موضح في المثال أدناه unknown سيمنع تايب سكربت:

```
let w: unknown = 1;
w = "string"; // no error
w = {
  runANonExistentMethod: () => {
    console.log("I think therefore I am");
  }
} as { runANonExistentMethod: () => void}
// How can we avoid the error for the code commented out below when
we don't know the type?
// w.runANonExistentMethod(); // Error: Object is of
type 'unknown'.
if(typeof w === 'object' && w !== null) {
  (w as { runANonExistentMethod:
Function }).runANonExistentMethod();
}
// Although we have to cast multiple times we can do a check in the
if to secure our type and have a safer casting
```

قارن المثال أعلاه بالمثال السابق، مع any.

من الأفضل استخدامه عندما لا تعرف نوع البيانات التي يتم كتابتها. لإضافة نوع لاحقًا، ستحتاج إلى إرساله unknown.

يتم الإرسال عندما نستخدم الكلمة المحجوزه "as" لنقول أن الخاصية أو المتغير من النوع المصبوب.

احذر هذا الخطا

يلقي خطأ بشكل فعال كلما تم تعريفه never.

```
let x: never = true; // Error: Type 'boolean' is not assignable to
type 'never'.
```

never نادرا ما يستخدم، خاصة في حد ذاته، استخدامه الأساسي هو في المعالجات الجنيسة المتقدمة.

النوع: الغير محدد

undefined وهي null جافاسكربت الأنواع التي تشير إلى أساسيات undefined وعلى null التوالي.

```
let y: undefined = undefined;
let z: null = null;
```

tsconfig.json الملف strictNullChecks تم تمكينها في هذه الأنواع ليس لها فائدة كبيرة إلا إذا.

مصفوفات تايب سكربت

يحتوي تايب سكربت على بناء جملة محدد لكتابة المصفوفات.

اقرأ المزيد عن المصفوفات في كتاب نود جى اس او جافاسكربت Array .

مثال

```
const names: string[] = [];
names.push("Dylan"); // no error
// names.push(3); // Error: Argument of type 'number' is not
assignable to parameter of type 'string'.
```

جعل المتغير للقراءة فقط

readonly يمكن أن تمنع الكلمة المحجوزه تغيير المصفوفات.

مثال

```
const names: readonly string[] = ["Dylan"];
names.push("Jack"); // Error: Property 'push' does not exist on
type 'readonly string[]'.
// try removing the readonly modifier and see if it works?
```

امكانية الاستنساخ

يمكن فى **تايب سكربت** استنتاج نوع المصفوفة إذا كانت تحتوي على قيم.

مثال

```
const numbers = [1, 2, 3]; // inferred to type number[]
numbers.push(4); // no error
// comment line below out to see the successful assignment
numbers.push("2"); // Error: Argument of type 'string' is not
assignable to parameter of type 'number'.
let head: number = numbers[0]; // no error
```

مصفوفة تايب سكربت

المصفوفات المكتوبة

الصف عبارة عن مصفوفة مكتوبة بطول وأنواع محددة مسبقًا لكل فهرس.

تعتبر المصفوفة رائعة لأنها تسمح لكل عنصر في المصفوفة بأن يكون نوعًا معروفًا من القيمة.

لتعريف صف، حدد نوع كل عنصر في المصفوفة:

مثال

```
// define our tuple
let ourTuple: [number, boolean, string];

// initialize correctly
ourTuple = [5, false, 'Coding God was here'];
```

كما ترون لدينا رقم، منطقي ونص . ولكن ماذا يحدث إذا حاولنا وضعها بالترتيب الخاطئ:

مثال

```
// define our tuple
let ourTuple: [number, boolean, string];

// initialized incorrectly which throws an error
ourTuple = [false, 'Coding God was mistaken', 5];
```

مهم في صفنا وسوف يؤدي إلى حدوث خطأ number والترتيب string، boolean على الرغم من أن لدينا.

اجعلها للقراءة فقط

الممارسة الجيدة هي أن تجعل صفك readonly .

تحتوي Tuples فقط على أنواع محددة بقوة للقيم الأولية:

مثال

```
// define our tuple
let ourTuple: [number, boolean, string];
// initialize correctly
ourTuple = [5, false, 'Coding God was here'];
// We have no type safety in our tuple for indexes 3+
ourTuple.push('Something new and wrong');
console.log(ourTuple);
```

فقط على أنواع محددة بقوة للقيم الأولية **Tuples** ترى أن القيمة الجديدة تحتوي:

مثال

```
// define our readonly tuple
const ourReadonlyTuple: readonly [number, boolean, string] =
[5, true, 'The Real Coding God'];
// throws error as it is readonly.
ourReadonlyTuple.push('Coding God took a day off');
```

من قبل قبل أن تتعامل مع المصفوفة على الأرجح **React** إذا كنت قد استخدمت.

`useState` تقوم بإرجاع صف من القيمة ووظيفة الضبط.

`const [firstName, setFirstName] = useState('Dylan')` هو مثال شائع.

والقيمة الثانية `string` a بسبب البنية، نعلم أن القيمة الأولى في قائمتنا ستكون من نوع قيمة معين في هذه الحالة `function` a.

Tuples

تسمح لنا المجموعات المُسمّاة بتوفير سياق لقيمنا في كل فهرس.

مثال

```
const graph: [x: number, y: number] = [55.2, 41.3];
```

توفر المجموعات المعلن عن قيمتها مزيدًا من السياق لما تمثله قيم الفهرس الخاصة بنا.

حذف المصفوفة

نظرًا لأن المصفوفة عبارة عن مصفوفات، فيمكننا أيضًا حذفها.

مثال

```
const graph: [number, number] = [55.2, 41.3];
const [x, y] = graph;
```

لمراجعة الحذف التحقق من ذلك هنا .

أنواع كائنات تايب سكربت

يحتوي **تايب سكربت** على بناء جملة محدد لكتابة الكائنات.

اقرأ المزيد عن الكائنات في فصل كائنات **جافاسكربت** .

مثال

```
const car: { type: string, model: string, year: number } = {
  type: "Toyota",
  model: "Corolla",
  year: 2009
};
```

يمكن أيضًا كتابة أنواع الكائنات مثل هذه بشكل منفصل، وحتى إعادة استخدامها.

الاستنساخ

يمكن فى **تايب سكربت** استنتاج أنواع الخصائص بناءً على قيمها.

مثال

```
const car = {
  type: "Toyota",
};
car.type = "Ford"; // no error
car.type = 2; // Error: Type 'number' is not assignable to type
'string'.
```

خصائص اختيارية

الخصائص الاختيارية هي خصائص لا يلزم تعريفها في تعريف الكائن.

مثال بدون خاصية اختيارية

```
const car: { type: string, mileage: number } = { // Error: Property
'mileage' is missing in type '{ type: string; }' but required in type
'{ type: string; mileage: number; }'.
  type: "Toyota",
};
car.mileage = 2000;
```

مثال مع خاصية اختيارية

```
const car: { type: string, mileage?: number } = { // no error
  type: "Toyota"
};
car.mileage = 2000;
```

توقيعات الفهارس

يمكن استخدام توقيعات الفهرس للكائنات التي لا تحتوي على قائمة محددة من الخصائص.

مثال

```
const nameAgeMap: { [index: string]: number } = {};
nameAgeMap.Jack = 25; // no error
nameAgeMap.Mark = "Fifty"; // Error: Type 'string' is not
assignable to type 'number'.
```

يمكن أيضًا التعبير عن توقيعات الفهرس مثل هذا باستخدام أنواع الأدوات المساعدة مثل `Record<string, number>`.

الاعداد فى تايب سكربت

التعداد هو " فئة " خاصة تمثل مجموعة من الثوابت (متغيرات غير قابلة للتغيير).

Enums تأتي في نكهتين `string` و`numeric`. لنبدأ بالرقم.

الاعداد الرقمية

افتراضيًا، سيقوم التعداد بتهيئة القيمة الأولى `0` وإضافة 1 إلى كل قيمة إضافية:

مثال

```
enum CardinalDirections {
  North,
  East,
  South,
  West
}
let currentDirection = CardinalDirections.North;
// logs 0
console.log(currentDirection);
// throws error as 'North' is not a valid enum
currentDirection = 'North'; // Error: "North" is not assignable to
type 'CardinalDirections'.
```

الاعداد الرقمية – والقيمة التلقائية

يمكنك تعيين قيمة التعداد الرقمي الأول وزيادة قيمته تلقائيًا من ذلك:

مثال

```
enum CardinalDirections {
  North = 1,
  East,
  South,
  West
}
// logs 1
console.log(CardinalDirections.North);
// logs 4
console.log(CardinalDirections.West);
```

الاعداد الرقمية – المهيأة بالكامل

يمكنك تعيين قيم أرقام فريدة لكل قيمة تعداد. ثم لن تتم زيادة القيم تلقائيًا:

مثال

```
enum StatusCodes {
  NotFound = 404,
  Success = 200,
  Accepted = 202,
  BadRequest = 400
}
// logs 404
console.log(StatusCodes.NotFound);
// logs 200
console.log(StatusCodes.Success);
```

الاعداد فى النص

يمكن أن تحتوي الاعداد أيضًا على ملفات **strings**. وهذا أكثر شيوعًا من الاعداد الرقمية، نظرًا لسهولة قراءتها وهدفها.

مثال

```
enum CardinalDirections {
  North = 'North',
  East = "East",
  South = "South",
  West = "West"
};
// logs "North"
console.log(CardinalDirections.North);
// logs "West"
console.log(CardinalDirections.West);
```

من الناحية الفنية، يمكنك مزج ومطابقة قيم التعداد التسلسلي والرقمي، ولكن يوصى بعدم القيام بذلك.

الاسماء المختصرة والكائنات لانواع تايب سكربت

يسمح **تايب سكربت** بتعريف الأنواع بشكل منفصل عن المتغيرات التي تستخدمها.

تسمح الاسماء المختصرة والكائنات بمشاركة الأنواع بسهولة بين المتغيرات/الكائنات المختلفة.

كيف اكتب الاسماء المختصرة

تسمح الاسماء المختصرة للنوع بتعريف الأنواع باسم مخصص (اسم مستعار).

يمكن استخدام الاسماء المختصرة للنوع مع الأنواع الأولية مثل string أو الأنواع الأكثر تعقيدًا مثل objects و arrays:

مثال

```
type CarYear = number
type CarType = string
type CarModel = string
type Car = {
  year: CarYear,
  type: CarType,
  model: CarModel
}

const carYear: CarYear = 2001
const carType: CarType = "Toyota"
const carModel: CarModel = "Corolla"
const car: Car = {
  year: carYear,
  type: carType,
  model: carModel
};
```

الكائنات

تشبه الكائنات الاسماء المختصرة للكتابة، إلا أنها تنطبق على الأنواع فقط object .

مثال

```
interface Rectangle {
  height: number,
  width: number
}
```

```
const rectangle: Rectangle = {
  height: 20,
  width: 10
};
```

توسيع الكائنات

يمكن للكائنات توسيع تعريف بعضها البعض.

توسيع الكائن يعني أنك تقوم بإنشاء كائن جديدة بنفس خصائص الكائن الأصلية، بالإضافة إلى شيء جديد.

مثال

```
interface Rectangle {
  height: number,
  width: number
}

interface ColoredRectangle extends Rectangle {
  color: string
}

const coloredRectangle: ColoredRectangle = {
  height: 20,
  width: 10,
  color: "red"
};
```

أنواع الانضمام

يتم استخدام أنواع الانضمام عندما تكون القيمة أكثر من نوع واحد.

مثل عندما تكون الخاصية `number` أو `string`.

يتم الانضمام بهذة العلامة | ومعناها(أو)

نقول أن المعلمة لدينا هي | باستخدام string أو number:

مثال

```
function printStatusCode(code: string | number) {
  console.log(`My status code is ${code}.`)
}
printStatusCode(404);
printStatusCode('404');
```

أخطاء نوع الانضمام

ملحوظة: يجب أن تعرف نوعك الذى تريدة عند استخدام الأنواع الموحدة لتجنب أخطاء الكتابة:

مثال

```
function printStatusCode(code: string | number) {
  console.log(`My status code is ${code.toUpperCase()}.`) // error:
Property 'toUpperCase' does not exist ontype 'string | number'.
  Property 'toUpperCase' does not exist on type 'number'
}
```

في مثالنا، نواجه مشكلة toUpperCase() في استدعاء الطريقة string ولا number يمكننا الوصول إليها.

وظائف تايب سكريبت

يحتوي تايب سكربت على بناء جملة محدد لكتابة معلمات الوظائف وقيم الإرجاع.

اقرأ المزيد عن الوظائف هنا .

نوع الإرجاع

يمكن تحديد نوع القيمة التي ترجعها الدالة بشكل صريح.

مثال

```
// the `: number` here specifies that this function returns a number
function getTime(): number {
  return new Date().getTime();
}
```

إذا لم يتم تحديد نوع الإرجاع، فسيحاول **تايب سكربت** استنتاجه من خلال أنواع المتغيرات أو التعبيرات التي تم إرجاعها.

نوع الإرجاع الفارغ فى الوظائف

يمكن استخدام النوع `void` للإشارة إلى أن الوظيفة لا تُرجع أي قيمة.

مثال

```
function printHello(): void {
  console.log('Hello!');
}
```

تتم كتابة معلمات الدالة بصيغة مشابهة لإعلانات المتغيرات.

مثال

```
function multiply(a: number, b: number) {
  return a * b;
}
```

ما لم تتوفر معلومات إضافية عن النوع كما هو ، any إذا لم يتم تحديد نوع معلمة، فسيتم استخدام **تايب سكربت** افتراضيًا
موضح في قسمي الكائنات الافتراضية والاسم المختصر للنوع أدناه.

الكائنات الاختيارية فى الاجرائات

تايب سكربت أن جميع الكائنات مطلوبة، ولكن يمكن وضع علامة عليها بشكل صريح على أنها اختيارية افتراضيًا، سيفترض.

مثال

```
// the `?` operator here marks parameter `c` as optional
function add(a: number, b: number, c?: number) {
  return a + b + (c || 0);
}
```

الكائنات الافتراضية فى الاجرائات

بالنسبة للمعلمات ذات القيم الافتراضية، تأتي القيمة الافتراضية بعد التعليق التوضيحي للنوع:

مثال

```
function pow(value: number, exponent: number = 10) {
  return value ** exponent;
}
```

تايب سكربت أيضًا استنتاج النوع من القيمة الافتراضية فى يمكن.

الكائنات المعلن عن قيمتها فى الاجرائات

تتبع كتابة الكائنات المعلن عن قيمتها نفس نمط كتابة الكائنات العادية.

مثال

```
function divide({ dividend, divisor }: { dividend: number, divisor:
number }) {
  return dividend / divisor;
}
```

الكائنات البسيطة فى الاجرائات

يمكن كتابة الكائنات البسيطة مثل الكائنات العادية، ولكن يجب أن يكون النوع مصفوفة لأن الكائنات البسيطة تكون دائمًا مصفوفات.

مثال

```
function add(a: number, b: number, ...rest: number[]) {
  return a + b + rest.reduce((p, c) => p + c, 0);
}
```

اكتب الاسم المختصر

يمكن تحديد أنواع الوظائف بشكل منفصل عن الوظائف ذات الاسماء المختصرة للأنواع.

تتم كتابة هذه الأنواع بشكل مشابه لوظائف الأسهم، اقرأ المزيد عن وظائف الأسهم هنا .

مثال

```
type Negate = (value: number) => number;

// in this function, the parameter `value` automatically gets
assigned the type `number` from the type `Negate`
const negateFunction: Negate = (value) => value * -1;
```

مفهوم الصب او الفرع

هناك أوقات عند العمل مع الأنواع حيث يكون من الضروري تجاوز نوع المتغير، كما هو الحال عندما يتم توفير أنواع غير صحيحة بواسطة المكتبة.

الصب هو عملية تجاوز النوع.

استخدام الكلمة as

هناك طريقة مباشرة لإخراج متغير وهي استخدام الكلمة as المحجوزه ، والتي ستغير نوع المتغير المحدد مباشرة.

مثال

```
let x: unknown = 'hello';
console.log((x as string).length);
```

لا يؤدي الإرسال فعليًا إلى تغيير نوع البيانات داخل المتغير، على سبيل المثال، لا تعمل التعليمات البرمجية التالية كما هو متوقع نظرًا لأن المتغير x لا يزال يحمل رقمًا.

```
let x: unknown = 4;
console.log((x as string).length); // prints undefined since
numbers don't have a length
```

سيستمر **تايب سكربت** في محاولة التحقق من عمليات الإرسال لمنع عمليات الإرسال التي لا تبدو صحيحة، على سبيل المثال، سيؤدي ما يلي إلى ظهور خطأ في الكتابة نظرًا لأن **تايب سكربت** يعرف أن تحويل نص إلى رقم ليس له معنى دون تحويل البيانات:

```
console.log((4 as string).length); // Error: Conversion of type
'number' to type 'string' may be a mistake because neither type
sufficiently overlaps with the other. If this was intentional,
convert the expression to 'unknown' first.
```

يغطي قسم فرض القوة أدناه كيفية تجاوز ذلك.

استخدام <>

استخدام <> يعمل بنفس طريقة الإرسال باستخدام as.

مثال

```
let x: unknown = 'hello';
console.log((<string>x).length);
```

كما هو الحال عند العمل على ملفات TSX، لن يعمل هذا النوع من الإرسال مع React.

Unknown | استخدام

لتجاوز أخطاء الكتابة التي قد يرتكبها تايب سكربت عند الإرسال، قم أولاً بالإرسال إلى unknown، ثم إلى النوع الهدف.

مثال

```
let x = 'hello';
console.log(((x as unknown) as number).length); // x is not
actually a number so this will return undefined
```

فئات او كلاسات تايب سكريبت

يضيف تايب سكربت الأنواع ومعدلات الرؤية إلى فئات او كلاسات جافاسكربت.

الأعضاء:و ألانواع

تتم كتابة أعضاء الفئة (الخصائص والأساليب) باستخدام التعليقات التوضيحية للنوع، على غرار المتغيرات.

مثال

```
class Person {
  name: string;
}

const person = new Person();
person.name = "Jane";
```

الأعضاء: المرئية او العامة فى الكلاس

يتم أيضًا منح أعضاء الكلاس معدّلات خاصة تؤثر على الرؤية.

هناك ثلاثة معدلات رئيسية للرؤية في تايب سكربت.

- public- (افتراضي) يسمح بالوصول إلى عضو الكلاس من أي مكان
- private- يسمح فقط بالوصول إلى عضو الكلاس من داخل الكلاس
- protected- يسمح بالوصول إلى عضو الكلاس من نفسه ومن أي فئات او كلاسات ترثه، وهو ما يتم تناوله في قسم الوراثة أدناه

مثال

```
class Person {
  private name: string;
```

```
  public constructor(name: string) {
    this.name = name;
  }

  public getName(): string {
    return this.name;
  }
}

const person = new Person("Jane");
console.log(person.getName()); // person.name isn't accessible from
outside the class since it's private
```

. هنا this المحجوزه الموجودة في الكلاس عادةً إلى مثيل الكلاس. اقرأ المزيد عنthis تشير الكلمة

خصائص الكلاسات

يوفر **تايب سكربت** طريقة مناسبة لتعريف أعضاء الفئة في المُنشئ، وذلك عن طريق إضافة مُعدِّلات الرؤية إلى المعلمة.

مثال

```
class Person {
  // name is a private member variable
  public constructor(private name: string) {}

  public getName(): string {
    return this.name;
  }
}

const person = new Person("Jane");
console.log(person.getName());
```

اجعلها للقراءة فقط

يمكن للكلمة المحجوزه readonly أن تمنع تغيير أعضاء الكلاس، كما هو الحال مع المصفوفات.

مثال

```
class Person {
  private readonly name: string;

  public constructor(name: string) {
    // name cannot be changed after this initial definition, which
has to be either at it's declaration or in the constructor.
    this.name = name;
  }

  public getName(): string {
    return this.name;
  }
}

const person = new Person("Jane");
console.log(person.getName());
```

الوراثة

الكلمة implements يمكن استخدام الكائنات (المغطاة هنا) لتحديد النوع الذي يجب أن يتبعه الكلاس من خلال المحجوزه .

مثال

```
interface Shape {
  getArea: () => number;
}

class Rectangle implements Shape {
  public constructor(protected readonly width:
number, protected readonly height: number) {}
```

```
  public getArea(): number {
    return this.width * this.height;
  }
}
```

مفصولة بفاصلة كما ، implements يمكن للفصل تنفيذ واجهات متعددة عن طريق إدراج كل واحدة منها بعد ذلك
يلي:class Rectangle implements Shape, Colored {

تمديد الوراثة:

الكلمة المحجوزه . يمكن للفئة أن تمتد لفئة واحدة فقطextends يمكن للفصول تمديد بعضها البعض من خلال.

مثال

```
interface Shape {
  getArea: () => number;
}

class Rectangle implements Shape {
  public constructor(protected readonly width:
number, protected readonly height: number) {}

  public getArea(): number {
    return this.width * this.height;
  }
}

class Square extends Rectangle {
  public constructor(width: number) {
    super(width, width);
  }

  // getArea gets inherited from Rectangle
}
```

مفهوم التجاوز

عندما يقوم الكلاس بتوسيع فصل آخر، يمكنه استبدال أعضاء الكلاس الأصلي بنفس الاسم.

الكلمة المحجوزه override تسمح الإصدارات الأحدث من **تايب سكربت** بوضع علامة واضحة على ذلك باستخدام .

مثال

```
interface Shape {
  getArea: () => number;
}

class Rectangle implements Shape {
  // using protected for these members allows access from classes
that extend from this class, such as Square
  public constructor(protected readonly width:
number, protected readonly height: number) {}

  public getArea(): number {
    return this.width * this.height;
  }

  public toString(): string {
    return `Rectangle[width=${this.width}, height=${this.height}]`;
  }
}

class Square extends Rectangle {
  public constructor(width: number) {
    super(width, width);
  }

  // this toString replaces the toString from Rectangle
  public override toString(): string {
    return `Square[width=${this.width}]`;
  }
}
```

تكون الكلمة المحجوزه override اختيارية عند تجاوز أسلوب ما، وتساعد فقط على منع التجاوز غير ،بشكل افتراضي noImplicitOverride المقصود لأسلوب غير موجود. استخدم الإعداد لفرض استخدامه عند التجاوز.

ملوحظة

يمكن كتابة الفئات او كلاسات بطريقة تسمح باستخدامها كفئة أساسية للفئات او كلاسات الأخرى دون الحاجة إلى تنفيذ جميع الأعضاء. ويتم ذلك باستخدام الكلمة المحجوزه **abstract** . الأعضاء الذين يتم تركهم دون تنفيذ يستخدمون الكلمة المحجوزه **abstract** أيضًا .

مثال

```
abstract class Polygon {
  public abstract getArea(): number;

  public toString(): string {
    return `Polygon[area=${this.getArea()}]`;
  }
}

class Rectangle extends Polygon {
  public constructor(protected readonly width:
number, protected readonly height: number) {
    super();
  }

  public getArea(): number {
    return this.width * this.height;
  }
}
```

لا يمكن إنشاء مثيل للفئات او كلاسات المجردة بشكل مباشر، حيث لم يتم تنفيذ جميع أعضائها.

مفهوم المعالجات العامة المحجوزه فى تايب سكربت

تسمح المعالجات العامة بإنشاء "متغيرات النوع" التي يمكن استخدامها لإنشاء فئات او كلاسات ووظائف وأسماء مختصرة لا تحتاج إلى تحديد الأنواع التي تستخدمها بشكل صريح.

تسهل المعالجات العامة كتابة تعليمات برمجية قابلة لإعادة الاستخدام.

المهام

تساعد المعالجات العامة ذات الوظائف في إنشاء طرق أكثر عمومية تمثل الأنواع المستخدمة والمرتجعة بشكل أكثر دقة.

مثال

```
function createPair<S, T>(v1: S, v2: T): [S, T] {
  return [v1, v2];
}
console.log(createPair<string, number>('hello', 42)); // ['hello',
42]
```

يمكن فى **تايب سكربت** أيضًا استنتاج نوع المعلمة العامة من معلمات الوظيفة.

الطبقات

يمكن استخدام المعالجات العامة لإنشاء فئات او كلاسات عامة، مثل **Map** .

مثال

```
class NamedValue<T> {
  private _value: T | undefined;

  constructor(private name: string) {}

  public setValue(value: T) {
    this._value = value;
  }

  public getValue(): T | undefined {
```

```
    return this._value;
  }

  public toString(): string {
    return `${this.name}: ${this._value}`;
  }
}

let value = new NamedValue<number>('myNumber');
value.setValue(10);
console.log(value.toString()); // myNumber: 10
```

يمكن فى **تايب سكربت** أيضًا استنتاج نوع المعلمة العامة إذا تم استخدامها في معلمة منشئة.

اكتب الاسماء المختصرة

تسمح المعالجات العامة الموجودة في الاسماء المختصرة للنوع بإنشاء أنواع أكثر قابلية لإعادة الاستخدام.

مثال

```
type Wrapped<T> = { value: T };

const wrappedValue: Wrapped<number> = { value: 10 };
```

يعمل هذا أيضًا مع الكائنات ذات الصيغة التالية:`interface Wrapped<T> {`

القيمة الافتراضية

يمكن تعيين قيم افتراضية للمعالجات العامة والتي تنطبق في حالة عدم تحديد أو استنتاج قيمة أخرى.

مثال

```
class NamedValue<T = string> {
  private _value: T | undefined;

  constructor(private name: string) {}

  public setValue(value: T) {
    this._value = value;
  }

  public getValue(): T | undefined {
    return this._value;
  }

  public toString(): string {
    return `${this.name}: ${this._value}`;
  }
}

let value = new NamedValue('myNumber');
value.setValue('myValue');
console.log(value.toString()); // myNumber: myValue
```

اضافة قيود

يمكن إضافة قيود إلى المعالجات العامة للحد من ما هو مسموح به. تتيح القيود إمكانية الاعتماد على نوع أكثر تحديدًا عند استخدام النوع العام.

مثال

```
function createLoggedPair<S extends string | number,
T extends string | number>(v1: S, v2: T): [S, T] {
  console.log(`creating pair: v1='${v1}', v2='${v2}'`);
  return [v1, v2];
}
```

يمكن دمج هذا مع القيمة الافتراضية.

أنواع الأدوات المساعدة فى تايب سكربت

يأتي **تايب سكربت** مزودًا بعدد كبير من الأنواع التي يمكن أن تساعد في بعض عمليات معالجة الكتابة الشائعة، والتي يشار إليها عادةً بالأنواع المساعدة.

يغطي هذا الكلاس أنواع المرافق الأكثر شعبية.

الخصائص

`Partial` يغير جميع الخصائص الموجودة في الكائن لتكون اختيارية.

مثال

```
interface Point {
  x: number;
  y: number;
}

let pointPart: Partial<Point> = {}; // `Partial` allows x and y to
be optional
pointPart.x = 10;
```

`Required` يغير كافة الخصائص في الكائن المطلوب.

مثال

```
interface Car {
  make: string;
  model: string;
  mileage?: number;
}
```

```
let myCar: Required<Car> = {
  make: 'Ford',
  model: 'Focus',
  mileage: 12000 // `Required` forces mileage to be defined
};
```

Recordهو اختصار لتعريف نوع الكائن بنوع مفتاح ونوع قيمة محددين.

مثال

```
const nameAgeMap: Record<string, number> = {
  'Alice': 21,
  'Bob': 25
};
```

Record<string, number>يعادل{ [key: string]: number }

حذف

Omitيزيل المفاتيح من نوع الكائن.

مثال

```
interface Person {
  name: string;
  age: number;
  location?: string;
}

const bob: Omit<Person, 'age' | 'location'> = {
  name: 'Bob'
```

```
  // `Omit` has removed age and location from the type and they
can't be defined here
};
```

Pick يزيل كافة المفاتيح باستثناء المفاتيح المحددة من نوع الكائن.

مثال

```
interface Person {
  name: string;
  age: number;
  location?: string;
}

const bob: Pick<Person, 'name'> = {
  name: 'Bob'
  // `Pick` has only kept name, so age and location were removed
from the type and they can't be defined here
};
```

Exclude يزيل الأنواع من الانضمام.

مثال

```
type Primitive = string | number | boolean
const value: Exclude<Primitive, string> = true; // a string cannot
be used here since Exclude removed it from the type.
```

نوع الإرجاع

ReturnTypeيستخرج نوع الإرجاع لنوع الوظيفة.

مثال

```
type PointGenerator = () => { x: number; y: number; };
const point: ReturnType<PointGenerator> = {
  x: 10,
  y: 20
};
```

Parametersيستخرج أنواع الكائنات من نوع الوظيفة كمصفوفة.

مثال

```
type PointPrinter = (p: { x: number; y: number; }) => void;
const point: Parameters<PointPrinter>[0] = {
  x: 10,
  y: 20
};
```

keyofهي كلمة أساسية في **تايب سكربت** تُستخدم لاستخراج نوع المفتاح من نوع الكائن.

Keyof مفاتيح

يتم إنشاء نوع موحد بهذه المفاتيح keyof، عند استخدامه على نوع كائن بمفاتيح صريحة.

مثال

```
interface Person {
  name: string;
  age: number;
}
// `keyof Person` here creates a union type of "name" and "age",
other strings will not be allowed
function printPersonProperty(person: Person, property: keyof
Person) {
  console.log(`Printing person property ${property}: "$
{person[property]}"`);
}
let person = {
  name: "Max",
  age: 27
};
printPersonProperty(person, "name"); // Printing person property
name: "Max"
```

توقيعات الفاهرس keyo

يمكن استخدامه أيضًا مع توقيعات الفهرس لاستخراج نوع الفهرس keyof.

```
type StringMap = { [key: string]: unknown };
// `keyof StringMap` resolves to `string` here
function createStringPair(property: keyof StringMap, value:
string): StringMap {
  return { [property]: value };
}
```

القيم undefined أو null لدى تايب سكربت نظام قوي للتعامل.

`strictNullChecks` يتم تعطيل المعالجة، ويمكن تمكينها عن طريق التعيين`undefined`، `null` بشكل افتراضي على **true.**

يتم تمكينها`strictNullChecks` تنطبق بقية هذه الصفحة عندما

`null`وهي`undefined` مثل الأنواع الأخرى مثل أنواع بدائية ويمكن استخدامها مثل `string`.

مثال

```
let value: string | undefined | null = null;
value = 'hello';
value = undefined;
```

تتم إضافتها بشكل صريح `undefined` تمكينه، يتطلب **تايب سكربت** تعيين القيم ما لم`strictNullChecks` عند إلى النوع.

التسلسل الاختياري

التسلسل الاختياري عبارة عن ميزة **جافاسكربت** تعمل بشكل جيد مع معالجة القيمة الخالية في **تايب سكربت**. فهو يسمح بالوصول إلى خصائص كائن ما، قد تكون موجودة أو لا تكون، باستخدام بناء جملة مضغوط. ويمكن استخدامه مع العلامة `.?`. المشغل عند الوصول إلى الخصائص

مثال

```
interface House {
  sqft: number;
  yard?: {
    sqft: number;
  };
}
function printYardSize(house: House) {
  const yardSize = house.yard?.sqft;
  if (yardSize === undefined) {
    console.log('No yard');
  } else {
```

```
    console.log(`Yard is ${yardSize} sqft`);
  }
}

let home: House = {
  sqft: 500
};

printYardSize(home); // Prints 'No yard'
```

التفريغ الاحطياطى

Nullish Coalescent هي إحدى ميزات **جافاسكربت** الأخرى التي تعمل أيضًا بشكل جيد مع المعالجة الخالية في **تايب سكربت**. يسمح بكتابة التعبيرات التي لها احتياطي على وجه التحديد عند التعامل مع `null` أو `undefined`. ويمكن استخدامه عامل التشغيل `??` مع عامل التشغيل `&&` في تعبير، على غرار استخدام.

مثال

```
function printMileage(mileage: number | null | undefined) {
  console.log(`Mileage: ${mileage ?? 'Not Available'}`);
}

printMileage(null); // Prints 'Mileage: Not Available'
printMileage(0); // Prints 'Mileage: 0'
```

كيف وضع القيمة الفارغة احطياطيا

نظام الاستدلال الخاص بـ **تايب سكربت** ليس مثاليًا، ففي بعض الأحيان يكون من المنطقي تجاهل احتمالية وجود `null` أو `undefined`. ! إحدى الطرق السهلة للقيام بذلك هي استخدام الإرسال، لكن **تايب سكربت** يوفر أيضًا القيمة عامل التشغيل كاختصار مناسب.

مثال

```
function getValue(): string | undefined {
  return 'hello';
}
let value = getValue();
console.log('value length: ' + value!.length);
```

تمامًا مثل عملية الصب، قد يكون هذا غير آمن ويجب استخدامه بحذر.

التعامل مع حدود المصفوفة

التمكين، سيفترض **تايب سكربت** افتراضيًا أن الوصول إلى المصفوفة لن يُرجع أبدًا غير `strictNullChecks` حتى مع محدد (ما لم يكن غير محدد جزءًا من نوع المصفوفة).

لتغيير هذا السلوك `noUncheckedIndexedAccess` يمكن استخدام التكوين.

مثال

```
let array: number[] = [1, 2, 3];
let value = array[0]; // with `noUncheckedIndexedAccess` this has
the type `number | undefined`
```