استایل کد

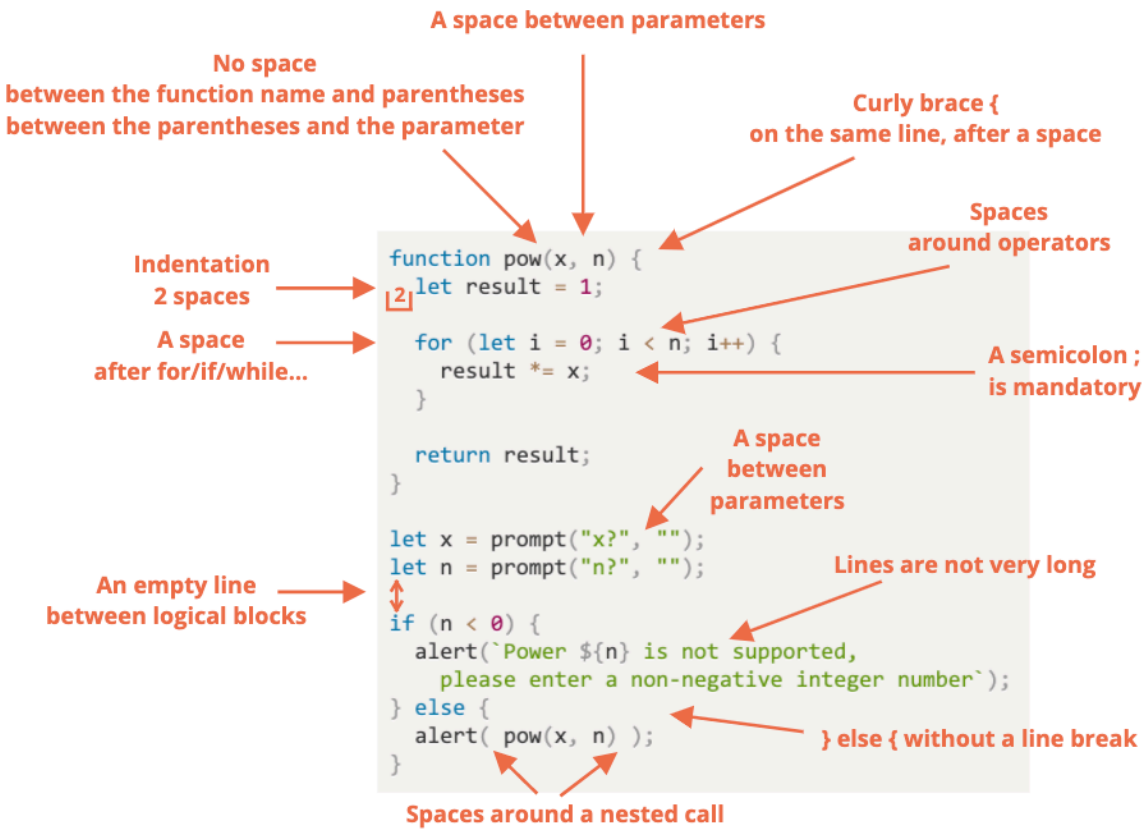
کد ما باید مرتب و تمیز بوده و خوانایی آن آسان باشد .

در واقع هنر واقعی برنامه نویسی این می باشد که تسک های پیچیده را طوری به کد تبدیل کنیم که هم صحیح باشد و هم خوانا باشد .

یک استایل خوب برای کد به این امر کمک شایانی خواهد کرد .

گرامر

در ادامه راهنمایی برای استایل کد را مشاهده می‌کنید که بعضی از قوانین پیشنهادی در آن گنجانده شده است .



در ادامه در مورد قوانین فوق و دلیل آنها توضیح خواهیم داد .

هیچ لزومی وجود ندارد که حتما از قوانین فوق پیروی کنید .

قوانین فوق فقط توصیه هایی برای نوشتن بهتر کد می‌باشد که هیچ بایدی در استفاده از آنها وجود ندارد .

آکولاد

در اکثر پروژه های جاوا اسکریپت ، آکولاد به مدل مصری نوشته می شود یعنی آکولاد اول در خط اول بدون اینکه در خط جدیدی نوشته شود قرار داده شده و قبل از آن یک فضای خالی قرار می گیرد .

```
if (condition) {
  // do this
  // ...and that
  // ...and that
}
```

ساختارهای یک خطی مانند if(condition) doSomething() قابل تامل هستند . آیا اصلا باید از آکولاد استفاده کنیم یا خیر؟

در ادامه چندین مثال آمده که به شما در تصمیم گیری کمک خواهد کرد :

- 😠 افراد مبتدی گاهی به این شکل کد می‌زنند : این نحوه نوشتن خوب نیست چرا که به آکولاد نیازی نمی‌باشد .

```
if (n < 0) {alert(`Power ${n} is not supported`);}
```

- 😠 جداسازی کدها در چند خط بدون استفاده از آکولاد . هیچ گاه این کار را نکنید چرا که بسیار مستعد خطا می‌باشد .

```
if (n < 0)
  alert(`Power ${n} is not supported`);
```

- 😏 نوشتن کدها در یک خط . در صورتی که کوتاه باشد قابل قبول است .

```
if (n < 0) alert(`Power ${n} is not supported`);
```

- 😃 بهترین حالت :

```
if (n < 0) {
  alert(`Power ${n} is not supported`);
}
```

برای کدهای کوتاه ، حالت یک خطی قابل قبول است ولی نوشتن کدها به شکل بلاکی و آخرین روش بالا ، خوانایی بیشتری دارد .

طول خط

هیچ کس دوست ندارد که خط افقی درازی را بخواند . بهتر است که خطوط طولانی به چند خط شکسته شوند .

برای مثال :

```
// backtick quotes ` allow to split the string into multiple lines
let str = `
  ECMA International's TC39 is a group of JavaScript developers,
  implementers, academics, and more, collaborating with the community
  to maintain and evolve the definition of JavaScript.
`;
```


و عبارت if را می‌توانیم به این شکل بنویسیم :

```
if (
  id === 123 &&
  moonPhase === 'Waning Gibbous' &&
  zodiacSign === 'Libra'
) {
  letTheSorceryBegin();
}
```

حداکثر طول خط باید به توافق اعضای تیم برسد ولی معمولا بیشتر از ۸۰ یا ۱۲۰ کاراکتر نمی‌شود.

indent (فاصله از ابتدای خط)

۲ نوع indent وجود دارد:

۱– افقی : ۲ یا ۴ space

ایندنت افقی از ۲ یا ۴ space یا یک تب (tab) تشکیل شده است. اینکه از کدام باید استفاده شود بحثی است که همیشه بین برنامه‌نویسان بوده است ولی امروزه معمولا از space استفاده می‌شود. یکی از مزایای space نسبت به tab این است که تنظیمات دلخواه بیشتری می‌توان روی آن انجام داد. برای مثال :

```
show(parameters,
    aligned, // 5 spaces padding at the left
    one,
    after,
    another
  ) {
  // ...
}
```

۲– عمودی : خطوط خالی برای تقسیم بندی منطقی کدها

حتی یک فانکشن هم می‌تواند به بخش‌های منطقی مختلفی تقسیم شود. در مثال زیر تعریف متغیر ، حلقه for و قسمت return به صورت عمودی از هم جدا شده است:

```
function pow(x, n) {
  let result = 1;
  //          <--
  for (let i = 0; i < n; i++) {
    result *= x;
  }
  //          <--
  return result;
```

```
}
```

برای خوانایی بیشتر کد ، هر جایی که نیاز است از خط خالی استفاده کنید . هیچگاه بیشتر از ۹ خط متوالی نباید بدون indent عمودی نوشته شوند .

سمی کالن

در انتهای همه خطوط باید از سمی کالن استفاده شود ، حتی جاهایی که می شود از آن چشم پوشی کرد .

زبان هایی وجود دارند که سیمی کالن در آنها واقعا اختیاری است و به ندرت استفاده می شود ولی در جاوا اسکریپت موقعیت هایی وجود دارد که در آنها خط جدید به معنای سمی کالن نبوده و کد را مستعد خطا می کند . در بخش های بعد در این مورد بیشتر توضیح خواهیم داد .

اگر در جاوا اسکریپت با تجربه باشید احتمالا از حالت بدون سمی کالن مانند StandardJS استفاده می‌کنید . در غیر این صورت بهتر است برای جلوگیری از بروز خطاهای احتمالی از سمی کالن استفاده کنید که اکثر برنامه نویسان نیز همین کار را می‌کنند .

تودرتویی

از نوشتن کد های عمیق تو در تو خودداری کنید .

برای مثال ، در حلقه ، بهتر است گاهی اوقات از continue استفاده کنید تا از نوشتن کدهای تو در توی بی مورد جلوگیری به عمل آید . برای مثال ، به جای اضافه کردن یک if تو در توی اضافه به شکل زیر :

```
for (let i = 0; i < 10; i++) {
  if (cond) {
    ... // <- one more nesting level
  }
}
```


بهتر است به این شکل بنویسیم :

```
for (let i = 0; i < 10; i++) {
  if (!cond) continue;
  ...  // <- no extra nesting level
}
```

به صورت مشابه نیز می توانیم با if/else و return برخورد کنیم.

برای مثال ، دو constructor زیر یکسان هستند :

مورد اول :

```
function pow(x, n) {
  if (n < 0) {
    alert("Negative 'n' not supported");
  } else {
    let result = 1;
    for (let i = 0; i < n; i++) {
```

```
      result *= x;
    }
    return result;
  }
}
```

مورد دوم :

```
function pow(x, n) {
  if (n < 0) {
    alert("Negative 'n' not supported");
    return;
  }
  let result = 1;
  for (let i = 0; i < n; i++) {
    result *= x;
  }
  return result;
}
```


مورد دوم خواناتر است چرا که حالت مخصوص n<0 در ابتدای کد قرار گرفته و تنها در صورتی وارد بدنه اصلی می شود که n مساوی یا بزرگتر 0 باشد بدون اینکه نیاز باشد بر پیچیدگی کد بیافزاییم.

محل قرارگیری فانکشن‌ها

اگر در کد خود چندین فانکشن دارید به سه طریق می‌توانید آنها را ساماندهی کنید :

۱– فانکشن ها را بالای کدهایی بنویسید که از آن استفاده می‌کنند :

```
// function declarations
function createElement() {
  ...
}
function setHandler(elem) {
  ...
}
function walkAround() {
  ...
}
```

```
// the code which uses them
let elem = createElement();
setHandler(elem);
walkAround();
```

۲– کدها را نوشته سپس فانکشن ها را اضافه کنید :

```
// the code which uses the functions
let elem = createElement();
setHandler(elem);
walkAround();

// --- helper functions ---
function createElement() {
  ...
}

function setHandler(elem) {
  ...
}

function walkAround() {
  ...
}
```


۳– ترکیب دو مورد بالا : فانکشن‌ها جایی تعریف شود که اولین بار از آن استفاده می‌شود.

در اکثر مواقع ، مورد دوم ترجیح داده می‌شود. به خاطر اینکه هنگام مطالعه کد ابتدا می‌خواهیم بدانیم که آن کد چه کاری انجام می‌دهد. اگر کد در ابتدا نوشته شود راحت‌تر متوجه این موضوع می‌شویم. ممکن است بعد از آن حتی به مطالعه فانکشن‌ها هم نیازی نداشته باشیم مخصوصا اگر نام فانکشن عملکرد آن را شرح دهد.

راهنمای استایل

راهنمای استایل قوانین کلی در مورد اینکه چگونه کد بزنیم را شامل می‌شود به عنوان مثال از چه کوتیشنی استفاده کنیم ، از چند space برای indent استفاده کنیم ، حداکثر طول خط چه قدر باشد و کلی موضوعات ریز دیگر.

وقتی همه اعضای تیم از این قواعد پیروی کنند ، کد نوشته شده یکدست دیده می‌شود و مهم نیست که کدام یک از اعضای تیم آن را نوشته است.

البته هر تیم می‌تواند استایل خودش را داشته باشد ولی معمولا نیازی به این کار نیست. راهنماهای زیادی وجود دارد که می‌توان یکی از آنها را انتخاب کرد.

بعضی از این راهنماها اینها هستند :

- Google JavaScript Style Guide
- Airbnb JavaScript StyleGuide
- Idiomatic.JS
- StandardJS
- and so on ...

اگر تازه برنامه نویسی را شروع کرده اید ، از تصویر ابتدایی این قسمت می‌توانید کمک بگیرید . سپس می‌توانید به سراغ باقی استایل‌ها بروید و تصمیم بگیرید که از کدام یک از آنها می‌خواهید استفاده نمایید .

لینترهای خودکار

لینترها ابزارهایی هستند که می‌توانند به صورت خودکار استایل کد را چک کرده و پیشنهادهایی برای بهبود وضعیت کد ارایه دهند .
مهمترین مزیت لینترها این است که این چک کردن خودکار استایل ، می‌تواند باگ‌هایی مانند خطاهایی تایپی در یک متغیر یا نام فانکشن‌ها را نیز پیدا کنند . به خاطر این ویژگی پیشنهاد می‌شود که همیشه از لینتر استفاده کنید حتی اگر احساس می کنید که نیاز ندارید تا از استایل خاصی پیروی کنید .
معروفترین لینترها عبارتند از :

JSLint : از اولین لینترها

JSHint :کارایی بیشتر از JSLint

ESLint : احتمالا جدیدترین لینتر موجود

هر سه اینها می‌توانند هدف مد نظر را انجام دهند . پیشنهاد ما ESLint است .

بیشتر لینترها همراه با ادیتورهای معروف وجود دارند فقط باید پلاگین مرتبط با آن را نصب کرده و تنظیمات دلخواه خود را انجام دهید .
برای مثال برای ESLint باید این مراحل را انجام دهید :

۱– نصب Node.js

۲– نصب ESLint توسط دستور npm install -g eslint

۳– ساخت فایل کانفیگ به نام .eslintrc در root پروژه جاوا اسکریپت (پوشه ای که همه فایل ها داخل آن قرار دارد)

۴– نصب یا فعال سازی پلاگین مورد نظر برای ادیتور مدنظرتان که ESLint را همراه خودش دارد . اکثر ادیتورهای معروف شامل آن می‌شوند .

مثال :

```
{
  "extends": "eslint:recommended",
  "env": {
    "browser": true,
    "node": true,
    "es6": true
  },
```

```
  "rules": {
    "no-console": 0,
    "indent": 2
  }
}
```

در این مثال عبارت extends مشخص می‌کند که تنظیمات لینتر بر اساس تنظیمات پیشنهاد لینتر بوده و بعد از آن تنظیمات دلخواه خودمان را اضافه کرده‌ایم.

همچنین می‌توانیم مجموعه‌ای از قوانین را از وب گرفته و به تنظیمات لینتر اضافه کنیم. برای اطلاعات بیشتر به آدرس زیر مراجعه نمایید :

http://eslint.org/docs/user-guide/getting-started

همچنین بعضی IDE ها لینتر داخلی خودشان را دارند که هر چند مناسب است ولی به قدرت ESLint نیست.

خلاصه

تمام قوانین ذکر شده در این بخش ، جهت افزایش خوانایی کد شما می‌باشد. در مورد هر یک از آنها می‌توان تجدید نظر کرد.

وقتی در مورد نوشتن کدهای بهتر فکر می‌کنیم باید از خودمان بپرسیم : چه چیزی کد را خواناتر و قابل فهم‌تر می‌کند؟ و چه چیزی می‌تواند جلوی بروز حظاها را بگیرد؟ این دو سوال را همیشه در خاطر داشته باشید.

همچنین مطالعه راهنماهای استایل معروف می‌تواند شما را به روز نگه داشته و با بهترین ایده‌ها و راه‌حل‌ها آشنا کند.