함수표현식 vs 함수선언식

함수 선언식 - Function Declarations

일반적인 프로그래밍 언어에서의 함수 선언과 비슷한 형식이다.

```
function 함수명() {
  구현 로직
}
```

```
// 예시 function funcDeclarations() {
  return 'A function declaration';
}
funcDeclarations();  // 'A function declaration'
```

함수 표현식 - Function Expressions

유연한 자바스크립트 언어의 특징을 활용한 선언 방식

```
var 함수명 = function () {
  구현 로직
};
```

```
// 예시 var funcExpression = function () {
    return 'A function expression';
}
funcExpression();  // 'A function expression'
```

함수 선언식과 표현식의 차이점

함수 선언식은 호이스팅에 영향을 받지만, 함수 표현식은 호이스팅에 영향을 받지 않는다.

함수 선언식은 코드를 구현한 위치와 관계없이 자바스크립트의 특징인 **호이스팅**에 따라 브라우저가 자바스크립트를 해석할 때 맨 위로 끌어 올려진다.

예를 들어, 아래의 코드를 실행할 때

```
// 실행 전 logMessage();
sumNumbers();

function logMessage() {
  return 'worked';
}

var sumNumbers = function () {
  return 10 + 20;
};
```

호이스팅에 의해 자바스크립트 해석기는 코드를 아래와 같이 인식한다.

```
// 실행 시 function logMessage() {
  return 'worked';
}

var sumNumbers;

logMessage();  // 'worked'sumNumbers();
               // Uncaught TypeError: sumNumbers is not a function

sumNumbers = function () {
  return 10 + 20;
};
```

위 코드 결과는 아래와 같다.

```
▶Uncaught TypeError: sumNumbers is not a function
```

함수 표현식 sumNumbers 에서 var 도 호이스팅이 적용되어 위치가 상단으로 끌어올려졌다.

```
var sumNumbers;

logMessage();
sumNumbers();
```

하지만 실제 sumNumbers 에 할당될 function 로직은 호출된 이후에 선언되므로, sumNumbers 는 함수로 인식하지 않고 변수로 인식한다.

호이스팅을 제대로 모르더라도 함수와 변수를 가급적 코드 상단부에서 선언하면, 호이스팅으로 인한 스코프 꼬임 현상은 방지할 수 있다.

함수 표현식의 장점

‘함수 표현식이 호이스팅에 영향을 받지 않는다’ 는 특징 이외에도 함수 선언식보다 유용하게 쓰이는 경우는 다음과 같다.

- 클로져로 사용
- 콜백으로 사용 (다른 함수의 인자로 넘길 수 있음)

함수 표현식으로 클로져 생성하기

클로져는 함수를 실행하기 전에 해당 함수에 변수를 넘기고 싶을 때 사용된다. 더 쉽게 이해하기 위해 아래 예제를 살펴보자.

```
function tabsHandler(index) {
    return function tabClickEvent(event) {
// 바깥 함수인 tabsHandler() 의 index 인자를 여기서 접근할 수 있다.
        console.log(index);// 탭을 클릭할 때 마다 해당 탭의 index 값을 표시};
}

var tabs = document.querySelectorAll('.tab');
var i;

for (i = 0; i < tabs.length; i += 1) {
    tabs[i].onclick = tabsHandler(i);
}
```

위 예제는 모든 .tab 요소에 클릭 이벤트를 추가하는 예제다. 주목할 점은 클로져를 이용해 tabClickEvent() 에서 바깥 함수 tabsHandler() 의 인자 값 index 를 접근했다는 점이다.

```
function tabsHandler(index) {
    return function tabClickEvent(event) {
        console.log(index);
    };
}
```

for 반복문의 실행이 끝난 후, 사용자가 tab 을 클릭했을 때 tabClickEvent() 가 실행된다. 만약 클로져를 쓰지 않았다면 모든 tab 의 index 값이 for 반복문의 마지막 값인 tabs.length 와 같다.

```
for (i = 0; i < tabs.length; i += 1) {
    tabs[i].onclick = tabsHandler(i);
```

```
}
```

클로져를 쓰지 않은 예제를 보자.

```
var tabs = document.querySelectorAll('.tab');
var i;

for (i = 0; i < tabs.length; i += 1) {
    tabs[i].onclick = function (event) {
      console.log(i);  // 어느 탭을 클릭해도 항상 tabs.length (i 의 최종 값) 이 출력
    };
}
```

위 소스는 탭이 3개라고 했을 때, 어느 탭을 클릭해도 i 는 for 반복문의 최종 값인 3이 찍힌다.

문제점을 더 파악하기 쉽게 for 문 안의 function() 을 밖으로 꺼내서 선언해보면

```
var tabs = document.querySelectorAll('.tab');
var i;
var logIndex = function (event) {
  console.log(i);  // 3
};

for (i = 0; i < tabs.length; i += 1) {
    tabs[i].onclick = logIndex;
}
```

logIndex 가 실행되는 시점은 이미 for 문의 실행이 모두 끝난 시점이다. 따라서, 어느 탭을 눌러도 for 문의 최종 값인 3이 찍힌다.

이 문제점을 해결하기 위해 클로져를 적용하면

```
function tabsHandler(index) {
    return function tabClickEvent(event) {
// 바깥 함수인 tabsHandler 의 index 인자를 여기서 접근할 수 있다.
        console.log(index);// 탭을 클릭할 때 마다 해당 탭의 index 값을 표시
    };
}

var tabs = document.querySelectorAll('.tab');
var i;

for (i = 0; i < tabs.length; i += 1) {
    tabs[i].onclick = tabsHandler(i);
}
```

for 반복문이 수행될 때 각 i 값을 tabsHandler() 에 넘기고, 클로져인 tabClickEvent() 에서 tabsHandler() 의 인자 값 index 를 접근할 수 있게 된다. 따라서, 우리가 원하는 각 탭의 index 를 접근할 수 있다.

함수 표현식을 다른 함수의 인자 값으로 넘기기

함수 표현식은 일반적으로 임시 변수에 저장하여 사용한다.

```
// doSth 이라는 임시 변수를 사용
var doSth = function () {
  // ...
};
```

함수 표현식을 임시 변수에 넣지 않고도 아래와 같이 콜백함수로 사용할 수 있다.

```
$(document).ready(function () {
  console.log('An anonymous function');// 'An anonymous function'});
```

jQuery 를 사용할 때 많이 보던 문법으로 위와 아래의 코드 결과는 같다.

```
var logMessage = function () {
  console.log('An anonymous function');
};

$(document).ready(logMessage);// 'An anonymous function'
```

자바스크립트 내장 API 인 forEach() 를 사용할 때도 콜백함수를 사용할 수 있다.

```
var arr = ["a", "b", "c"];
arr.forEach(function () {
  // ...
});
```

콜백 함수란 다른 함수의 인자로 전달된 함수를 의미합니다. 자바스크립트가 일급 객체로서 가지는 특징 중 하나죠. 자세한 내용은 여기를 참고하세요.

결론

함수 표현식이 선언식에 비해 가지는 장점이 많지만, 결국에는 이러한 차이점을 인지한 상태에서 일관된 코딩 컨벤션으로 코드를 작성하는 게 중요하다는 생각이 듭니다. AirBnb 의 JS

Style 가이드 에서도 함수 선언식보다는 함수 표현식을 지향하고 있는데요. 그래도 자기가 코딩하기 편한 방식으로 구현하는 게 좋지 않을까요?

본 글은 Site Point 의 Function Expressions vs Function Declarations 를 참고했습니다.