지금 까지는 함수 선언만 보았습니다. 함수를 선언하면 함수에 식별자(함수명)이 모두 주어집니다. 그러나 자바스크립트는 익명 함수도 지원합니다. 익명 함수라는 말은 식별자가 없다는 말인데, 도대체 어떻게 호출 할 수 있을까요 ? 이는 함수 표현 식이라는 것으로 가능하게 됩니다.

우리는 표현식이 값이 되고 , 함수 역시 값이 된다는 것을 알고 있습니다. 함수표현식은 함수를 선언하는 한가지 방법일 뿐이며, 그 함수가 익명이 될수도 있습니다. 함수표현식은 식별자에 할당할 수도 있고, 즉시 호출도 할 수 있습니다.

함수표현식은 함수 이름을 생략할 수 있다는 점을 제외하면 함수 선언과 문법적으로 완전히 같습니다.

간단하게 두 값을 더한 값을 반환하는 함수를 함수표현식으로 만들어 보도록 하겠습니다.

Const added = function(numb\_1, numb\_2){

Return numb\_1+numb\_2;

}

이 함수는 기본적으로 일반적인 함수와 동일합니다.

Function added(numb\_1,numb\_2){

…

}

여기서 부터는 지극히 이론적인 부분이며, 실제 개발에서 신경쓰며, 고민할 수 있는 부분은 아니기에, 조금 마음을 놓고 들으시기를 바라겠습니다.

이 두가지 함수 선언의 방식의 차이점은 먼저 함수 표현식으로 익명함수를 만들고 , 그 함수를 변수에 할당했다는 것입니다. 익명함수는 어디든지 사용할 수 있습니다. 다른 함수의 전달인자로 넘겨줄 수 있고, 객체의 함수 프로퍼티가 될 수 있습니다. 그렇다면 함수에 이름을 정의 하고 다시 변수에 할당하면 어떻게 될까요 ?

*const* a = *function* f(){

*console*.log('hello');

}

a();

f(); // 안됨

이렇게 작성하게 되면 함수 바깥에서는 식별자 f로 접근을 할 수 가 없고, a로 접근을 해야합니다. 함수에 두가지 이름을 붙이는게 좋은 것은 아니지만, 여기서 그렇게 한 이유는 이름 붙은 함수 표현식이 어떻게 동작하는지 명확하게 설명드리기 위함입니다. 이런식으로 함수를 만들고 함수 내부에서 자신을 호출하려면 함수의 이름으로 호출해야합니다. 함수 표현식이나, 함수 선언식이나 동일한데, 자바스크립트는 이 둘을 구분하여 , f내부에서는 f로 호출해야 자기 자신을 호출 할 수 있습니다. 이 답은 자바스크립트가 어디서 실행되어 주변환경을 파악하는 컨텍스트라는 개념을 이해해야 합니다. 현재는 이 차이에 대해서 생각할 필요가 없고, 단순히 나중에 호출할 생각으로 함수를 만든다면 함수 선언을 사용하면되고, 후에 나오는 콜백으로서 다른 함수의 파라메터로 함수를 넘길 예정이라면, 함수표현식을 사용하면됩니다.

이 익명함수를 조금 더 활용해보도록 하겠습니다. 이 익명함수를 조금 더 다이나믹하게 사용하는 방법은 단순히 함수의 파라메터로 함수를 넘기는 방법이 아니라, 익명함수라는 이름에 걸맞게 함수를 재 호출 할 수 없게 함수를 선언하자마자 호출하는 방법입니다.

*function* merryChrismas(){

*console*.log('hello');

(*function*(){

*console*.log("world");

})();

}

merryChrismas();

왜 이런 일을 하게되는 것일까요 ? 분명 함수의 목적은 재사용과 밀접한 관계가 있다고 했는데, 어째서 함수를 재 사용할 수 없도록 하는 것일까요 ?

이 “즉시 실행 함수”라는 녀석은 주로 한번의 실행만 필요로 하는 초기화 코드 부분에 많이 사용합니다. 그렇다면 왜 초기화 부분에서 많이 사용하게 되는 것일까요 ? 바로 변수를 전역으로 선언하는 것을 피하기 위해서 입니다. 전역에 변수를 추가하지 않아도 되기 때문에 코드 충돌 없이 구현이 가능합니다.

*let* initText;

(*function* (*number*) {

*const* textList = ["is Odd Text", "is Even Text"];

if (number % 2 == 0) {

initText = textList[1];

} else {

initText = textList[0];

}

})(5);

*console*.log(initText);

*console*.log(textList);

textList를 확인 해보면 당연하게도 textList가 저장되지 않고 initText만 초기화 된 것을 확인 할 수 있습니다.

만약 이를 기명함수로 만든다면 이 함수는 한번만 실행되어야하는 의도와 다르게 프로그래밍상 함수명의 충돌로 인한 재 호출, 콘솔창을 통한 호출, 등으로 프로그램이 잘못될 여지가 있습니다. 그러나 이런 익명함수를 사용하게 되면 이 함수는 호출과 동시에 재 호출될 수 없는 함수가 되게 되고 , 함수 내부에서 적용된 지역변수와 전역변수간 충돌이 없어집니다. 물론 지금은 이해하기 매우 어렵지만 점차 기능들을 구현하는 단계에서 더욱 익히게 될 것이니 걱정 마십셔

다음 함수를 만들기 위한 다른 방법은 화살표함수를 이용하는 것입니다.

이 화살표함수는 es6에서 새로 만든 문법으로 간단히 말해 function이라는 단어와 중괄호 숫자를 줄이려고 고안된 단축 문법입니다. 다만 한가지 중요한 차이가 는데, 이는 현재 배우지 않은 수준에서 설명해야 하기에 아주 작게만 언급하고, 후에 언급하도록 하겠습니다.

화살표 함수에는 세가지 단축 문법이 있습니다.

Fucntion을 생략해도 됩니다.

함수에 매개변수가 단 하나라면 ()괄호를 생략할 수 있습니다.

함수 바디가 표현식 하나라면 중괄호와 return문도 생략할 수 있습니다.

화살표 함수는 항상 익명이기에 함수표현식을 이용해서 함수를 만들어야 합니다.

Const f1 = () =>{

Console.log(‘its arrow function);

Console.log(‘hi’);

}

Const f2 = name =>{

Console.log(name);

}

Const f3 = () => console.log(‘hello world”);

이 화살표 함수의 가장 중요한 특징은 this가 다른 변수와 마찬가지로 정적으로 묶인다는 것입니다. 이 렉시컬 스코프라고 하는 것은 다음시간에 알아볼 스코프라는 개념에서 더욱 알아보도록 하겠습니다 .