

함수 생성

2022년 10월 4일 화요일 오후 8:24

[익명 함수]

Let 함수_이름 = function () {}

```
let 함수 = function(){
  console.log("함수 생성됨");
}
console.log(함수);
// undefined
// 함수 호출
함수();
// 함수 생성됨
```

화면 캡처: 2022-10-06 오후 5:50

[선언적 함수]

Function 함수_이름() {}

```
function 함수(){
  console.log("함수 생성됨");
}
// 함수 호출
함수();
// 함수 생성됨
// 함수 호출
함수();
// 함수 생성됨
// 함수 호출
함수();
// 함수 생성됨
```

[화살표 함수]

() => {}

```
let fn = () => {
  console.log("화살표 함수");
}
// 함수 호출
fn();
// 함수 생성됨
```

화면 캡처: 2022-10-06 오후 6:00

[함수의 기본형태]

Function 함수 이름 (매개변수) {
함수코드 ;
return 리턴 값 ;
}

```
> function power(x) {
  return x*x;
}
// 함수 호출
console.log(power(10));
// 100
// 함수 호출
console.log(power(20));
// 400
```

매개 변수가 여러 개인 경우

```
function multiply(a,b){
  return a*b;
}
// 함수 호출
console.log(multiply(2,3));
// 6
```

리턴 없는 함수

```
> function print(message){
  console.log(`${message}` 라 말했습니다);
}
// 함수 호출
print("안녕");
// 안녕
```

이게나오면 안돼

화살표 함수와 리턴

```
> let power = (x) => x * x;
// 함수 호출
console.log(power(10));
// 100
```

화살표 사용시 중괄호 작성하지 않음

[함수의 기본 활용 형태]

Function (매개변수, 매개변수) {
Let output = 초기값;
//output 계산
return output ;
}

```
> function sum(min, max){
  let output = 0;
  for(let i = min; i<=max; i++){
    output += i;
  }
  return output;
}
// 함수 호출
console.log(sum(1,100));
// 5050
```

화면 캡처: 2022-10-06 오후 6:13

[함수 매개 변수 초기화]

매개 변수 입력하지 않아도 함수가 호출된다

\$() 활용

[콜백 함수]

함수를 변수에 저장 할 수 있어 함수를 함수의 매개 변수로 전달 할 수 있다

이때 함수의 매개 변수로 전달되는 함수를 콜백 함수라고 한다.

```
> function callTenTimes(callback){
  for(let i =0; i<10; i++){
    callback();
  }
}
// 함수 호출
callTenTimes(function() {
  console.log("함수호출");
});
// 함수호출
// 함수호출
// 함수호출
// 함수호출
// 함수호출
// 함수호출
// 함수호출
// 함수호출
// 함수호출
// 함수호출
```

화면 캡처: 2022-10-06 오후 6:20

표준 내장 함수

2022년 10월 6일 목요일 오후 6:20

[숫자 변환 함수]

Number() 외에 더 있음

함수	설명
parseInt()	문자열을 정수로 변환
parseFloat()	문자열을 실수로 변환

```
> let inputA = "123";
< undefined
> let inputB = "12.223";
< undefined
> let inputC = "1401문";
< undefined
> console.log(parseInt(inputA));
52
< undefined
> console.log(parseInt(inputB));
52.223
< undefined
> console.log(parseInt(inputC));
< Uncaught SyntaxError: Unexpected token '문'
> console.log(parseFloat(inputC));
1401
< undefined
```

화면 캡처: 2022-10-06 오후 7:15

- Parseint("숫자",2); :: 2진법 표시

```
> parseInt("100",2);
< 4
> parseInt("1000",2);
< 8
```

화면 캡처: 2022-10-06 오후 7:17

- Parseint("숫자",8); :: 8진법 표시

```
> parseInt("100",8);
< 64
> parseInt("1000",8);
< 512
```

화면 캡처: 2022-10-06 오후 7:17

[숫자 생성 방법과 진수]

숫자 생성 방법	설명
0숫자	8진수 숫자를 만든다
숫자	10진수 숫자를 만든다
0x숫자	16진수 숫자 만든다.

```
> 010
< 8
> 0x10
< 16
```

[타이머 함수]

함수	설명
setTimeout(함수, 시간)	특정 시간 후에 함수를 실행
setInterval(함수, 시간)	특정 시간마다 함수실행
clearInterval(아이디)	특정 시간마다 실행했던 함수 호출 정지

```
> setTimeout(function() {
  console.log("1.2가 지났다");
}, 1000);
< 67
1.2가 지났다
> setInterval(function() {
  console.log("1.3 기다리다 호출");
}, 1000);
< 68
1.3 기다리다 호출
>
```

```
> let id = setInterval(function(){ console.log("종료"); }, 1000);
< undefined
> clearInterval(id);
< 69
종료
>
```

3초후에 종료

[익명 함수와 선언적 함수의 생성 순서]

1. 함수뒤어쓰기

Let 변수;
변수 = 10;
변수 = 20; -> 기존의 10 대신 20 덮어쓴다
Console.log(변수) //20

```
> let a;
< undefined
> a = function() { console.log("1변"); };
< f () { console.log("2변"); }
> f () { console.log("2변"); };
< f () { console.log("2변"); }
> g = function() { console.log("2변"); };
< f () { console.log("2변"); }
> g();
< Uncaught ReferenceError: g is not defined
at <anonymous>:1:1
> a();
2변
```

화면 캡처: 2022-10-06 오후 7:28

```
> a = function() { console.log("1변"); };
< f () { console.log("2변"); }
> function g() { console.log("2변"); };
< Uncaught SyntaxError: Identifier 'g' has already been declared
> function g() { console.log("2변"); };
< Uncaught SyntaxError: Identifier 'g' has already been declared
> g();
1변
```

-> 선언적 함수는 코드를 실행하기 전에 생성됨
선언적 함수가 먼저 생성되고 이후에 익명 함수를 만든다
그래서 1번 출력

익명함수는 선언적 함수를 무조건적으로 덮어쓴다.

2. 일반함수와 화살표 함수의 차이

//익명 함수 생성 후 곧바로 호출

```
(function() {
  console.log(this);
})();

window {fn: Window, window: Window, self: Window, document: document, name: "", location: Location, ...}
this {fn: Window, window: Window, self: Window, document: document, name: "", ...}
HTMLCanvasElement.prototype {f: t,e}
ShadowCSS {prepareTemplate: f, prepareTemplateStyles: f, prepareTemplateAlert: f alert(), ...}
atob: f atob()
btoa: f btoa()
caches {CacheStorage {}}
cancelAnimationFrame: f cancelAnimationFrame()
cancelIdleCallback: f cancelIdleCallback()
captureEvents: f captureEvents()
chrome {metricsPrivate: {}, loadTimes: f, csi: f}
clearInterval: f clearInterval()
clearTimeout: f clearTimeout()
clientInformation: Navigator {vendorSub: "", productSub: "20030302", ...}
close: f close()
closed: false
confirm: f confirm()
cookieStore: CookieStore {onchange: null}
cr: {onDidResume: f, onWillResume: f}
createImageBitmap: f createImageBitmap()
crossOriginIsolated: false
crypto {crypto: SubtleCrypto}
customElements {CustomElementRegistry {}}
defaultStatus: ""
devicePixelRatio: 1
document {document}
```

여기서 this 는 자바스크립트 최상위 객체 또는 외부에서 강제로 연결한 객체를 나타낸다

//화살표 함수 생성 후 곧바로 호출

```
{ () => {
  console.log(this);
}}();

window {fn: Window, window: Window, self: Window, document: document, name: "", location: Location, ...}
this {fn: Window, window: Window, self: Window, document: document, name: "", ...}
HTMLCanvasElement.prototype {f: t,e}
ShadowCSS {prepareTemplate: f, prepareTemplateStyles: f, prepareTemplateAlert: f alert(), ...}
atob: f atob()
btoa: f btoa()
caches {CacheStorage {}}
cancelAnimationFrame: f cancelAnimationFrame()
cancelIdleCallback: f cancelIdleCallback()
captureEvents: f captureEvents()
chrome {metricsPrivate: {}, loadTimes: f, csi: f}
clearInterval: f clearInterval()
clearTimeout: f clearTimeout()
clientInformation: Navigator {vendorSub: "", productSub: "20030302", ...}
close: f close()
closed: false
confirm: f confirm()
cookieStore: CookieStore {onchange: null}
cr: {onDidResume: f, onWillResume: f}
createImageBitmap: f createImageBitmap()
crossOriginIsolated: false
crypto {crypto: SubtleCrypto}
customElements {CustomElementRegistry {}}
defaultStatus: ""
devicePixelRatio: 1
document {document}
```

여기서 this 는 자기 자신과 관련된 것만 나타낸다