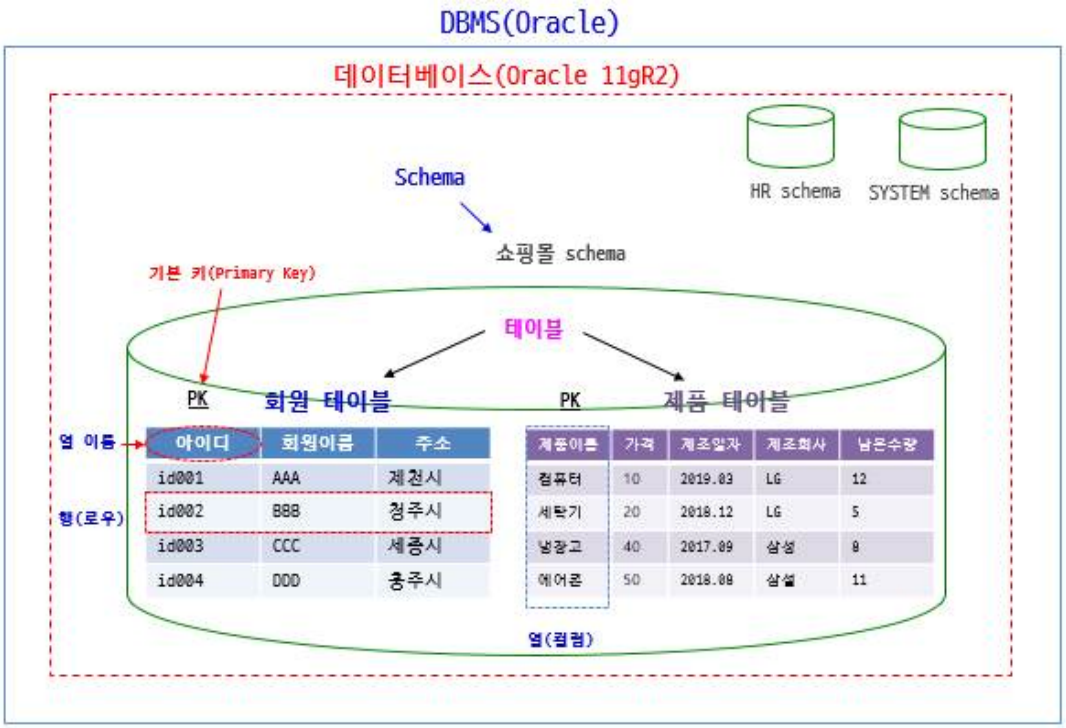
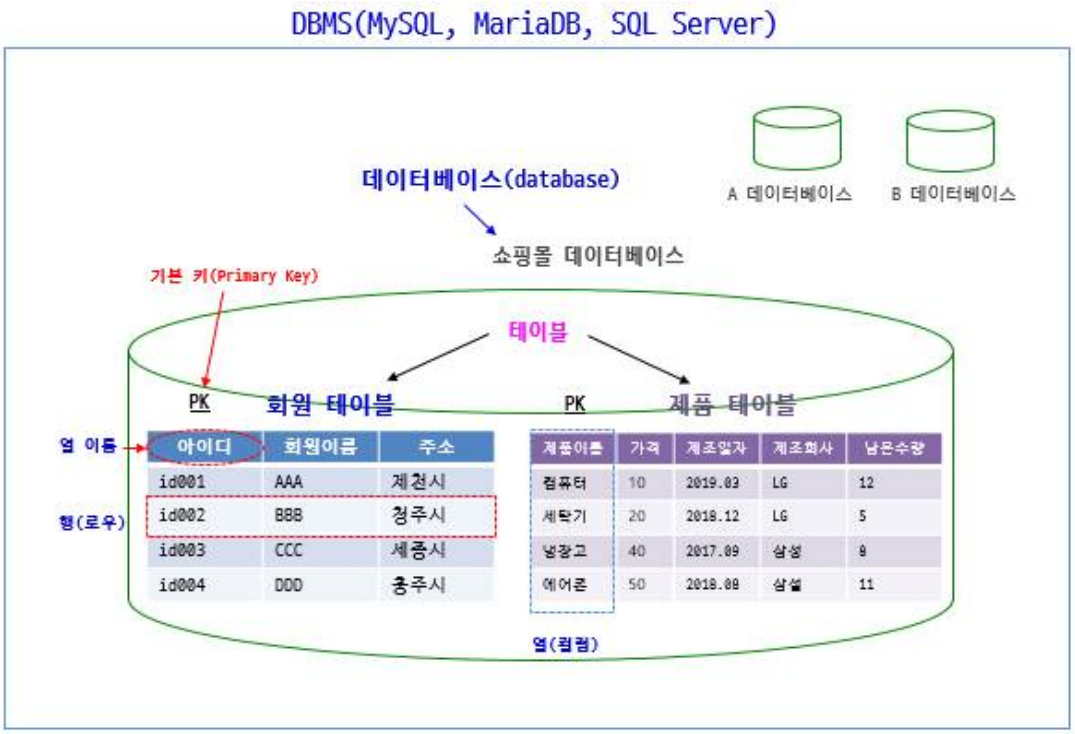
Computer Culture 에서 사용되는 용어 관계 정의

☑ DataBase Schema Architecture 및 용어 정의





☑ Realtional Data Model

❖ relation : row(행)과 Column(열)로 구성된 table

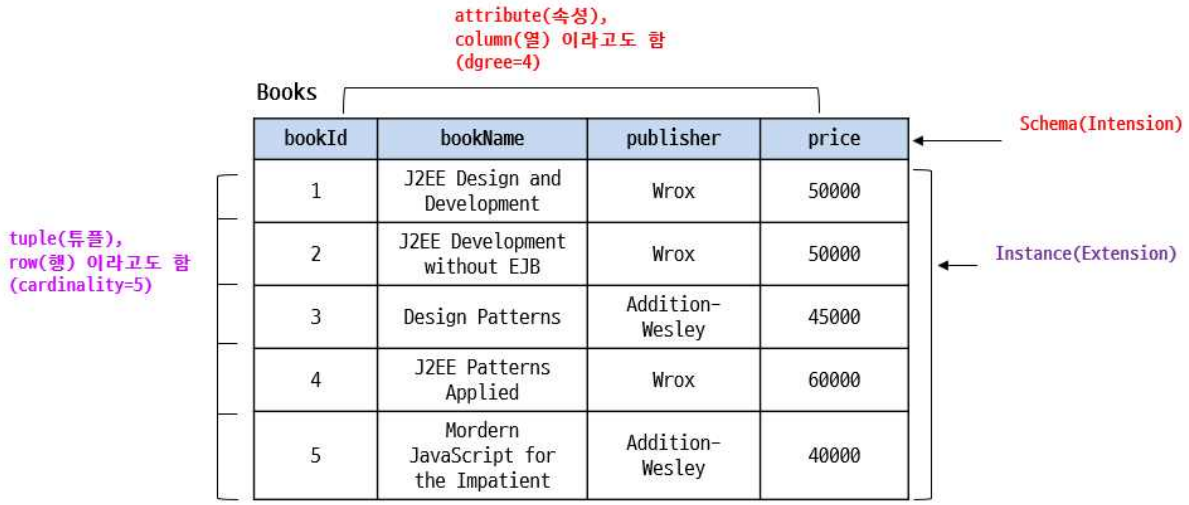
용어	한글 용어	비고
relation	릴레이션, table	"관계"라고 하지 않음
relational data model	관계 데이터 모델	
relational database	관계 데이터 베이스	
relational algebra	관계 대수	
relationaship	관계	

☑ Realtional Algebra Operator (관계 선형 대수)

연산자 종류	대상	연산자 이름			기호	설명
기본	단항	**셀렉션**			σ	릴레이션에서 조건에 만족하는 투플을 **선택**
기본	단항	**프로젝션**			π	릴레이션의 속성을 선택
추가	단항	개명			ρ	릴레이션이나 속성의 이름을 변경
유도	이항	디비전			$\div$	부모 릴레이션에 포함된 투플의 값을 모두 갖고 있는 투플을 분자 릴레이션에서 추출
기본	이항	**합집합**			$\cup$	두 릴레이션의 합집합
기본	이항	**차집합**			$-$	두 릴레이션의 차집합
유도	이항	교집합			$\cap$	두 릴레이션의 교집합
기본	이항	**카디전 프로덕트**			$\times$	두 릴레이션에 속한 모든 투플의 집합
유도	이항	조인	세타		$\bowtie$	c(θ) : 두 릴레이션 간의 비교 조건에 만족하는 집합
			동등		$\bowtie$	c(=) : 두 릴레이션 간의 같은 값을 가진 집합
			자연		$\bowtie_N$	동등 조인에서 중복 속성을 제거
			세미	left	$\ltimes$	자연 조인 후 오른쪽 속성을 제거
				right	$\rtimes$	자연 조인 후 왼쪽 속성을 제거
			외부	left	$⟕$	• 자연 조인 후 각각 왼쪽(left), 오른쪽(right), 양쪽(full)의 모든 값을 결과로 추출 • 조인이 실패(또는 값이 없을 경우)한 쪽의 값을 NULL로 채움
				right	$⟖$	
				full	$⟗$	

☑ Realtion Schema & Instance

- ☞ Relation Schema는 Relation(table)에 어떤 정보가 담길지 Definition(정의)한다.
- ☞ Schema는 Relational DataBase의 Relation(table)이 어떻게 구성되는지 어떤 정보를 담고 있는지에 대한 기본적인 구조를 Definition(정의)한다.
- ☞ table에서 schema는 table의 첫 행인 header에 나타나며 각 데이터의 특징을 나타내는 attribute, data type 등의 정보를 담고 있다.
- ☞ Instance는 정의된 schema에 따라 table에 실제로 저장되는 data의 set을 의미한다.



bookId	bookName	publisher	price
1	J2EE Design and Development	Wrox	50000
2	J2EE Development without EJB	Wrox	50000
3	Design Patterns	Addition-Wesley	45000
4	J2EE Patterns Applied	Wrox	60000
5	Mordern JavaScript for the Impatient	Addition-Wesley	40000

☑ Schema의 요소

- ☞ attribute : relation schema의 column(열)
- ☞ domain : attribute이 가질 수 있는 값의 set(집합)
- ☞ degre : attribute의 개수

☑ Programming Language 별 중요한 Data Type 용어 정의

Java (1995)	JavaScript (1995/2015)	Python (1991/2008)
Array : [동일한 data type] => Object	Array const smu = ["smart01", "ICT03", "ICT04"];	List : integer, string, real number.. mylist= ["smu01", "smu02", "smu03"]
Set 계열: 중복 불가능, 순서 없음. => Collection Framework	// Array Declaration 방법 1) let arr = new Array(); 2) let arr = [];	Set : 중복 불가능, 순서가 없음 myset= {"smu01", "smu02", "smu03"}
List 계열: 중복 가능, 순서 유지 => Collection Framework		Tuple : 수정 불가능, 읽기만 가능 myset= ("smu01", "smu02", "smu03")

Map 계열: key와 value 쌍으로 저장
=> key는 중복이 안됨
=> Collection Framework
 	
ex)
map.put("smart01", 20222240001);
map.put("smart02", 20222240002);
	Object
	const stu = {
	name: "smart01",
	sid: "20222240001",
	year: 2022
	}
		Dictionary => {key: value} 로 구성
		thisdict = {
		"brand": "smu",
		"school": "smart",
		"year": 2022
		}

[Java Data Types] ❖ Primitive Data Type byte short int long float double char boolean ❖ Reference Data Type class interface array	[Data Type & Data Structure] ❖ Primitive Values Number BigInt String Boolean Null Undefined Symbol ❖ Objects(Data Collection) object	[Python Data Type] ❖ Number Types int float complex ❖ Boolean Type bool ❖ Sequence Types str list tuple ❖ Mapping Type dict ❖ Set Type set ❖ 기타 Type function

☑ JSON

https://ko.wikipedia.org/wiki/JSON

JSON(JavaScript Object Notation)은 attribute-value pairs and array data types 또는 "key-value 쌍"으로 이루어진 data object를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 open standard file format(개방형 표준 포맷)이다. Asynchronous Browser/Server 통신(AJAX)을 위해, 넓게는 XML(AJAX가 사용)을 대체하는 주요 데이터 포맷이다. 특히, Internet에서 자료를 주고 받을 때 그 자료를 표현하는 방법으로 알려져 있다. 자료의 종류에 큰 제한은 없으며, 특히 컴퓨터 프로그램의 변수값을 표현하는 데 적합하다.

본래는 JavaScript 언어로부터 파생되어 JavaScript의 구문 형식을 따르지만 언어 독립형 data format이다. 즉, Programming Language나 Platform에 독립적이므로, 구문 분석 및 JSON 데이터 생성을 위한 코드는 C, C++, C#, Java, JavaScript, Perl, Python 등 수많은 프로그래밍 언어에서 쉽게 이용할 수 있다.

JSON foramt은 본래 Douglas Crockford가 규정하였다. RFC 7159와 ECMA-404 그리고 ISO/IEC 21778:2017 표준에 의해 기술되고 있다. KS 부합화 표준은 아직 제정되지 않았으며, TTA 협회 표준명은 TTAE.OT-10.0394 이다. ECMA 표준과 ISO/IEC 표준은 문법만 정의할 정도로 최소한으로만 정의되어 있는 반면 RFC는 시맨틱, 보안적 고려 사항을 일부 제공한다. JSON의 공식 인터넷 미디어 타입은 application/json이며, JSON의 파일 확장자는 .json이다.

☑ JSON Syntax Example

```
{
  "name": "SmartIT",
  "age": 01,
  "address": {
    "streetAddress": "65 Semyung-ro",
    "city": "Jecheon",
    "state": "Chungbuk",
    "postalCode": "27136"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "043-649-1234"
    },
    {
      "type": "office",
      "number": "043-649-1747"
    }
  ]
}
```

☑ JavaScript Function Declaration (함수 선언)

☞ function declaration 방식을 이용하면 function을 만들 수 있다. function keyword, function name, ()로 둘러싼 parameter를 차례로 써주면 function을 선언할 수 있다. 다음 함수는 parameter가 없는 함수이다.

```
function showMessage() {
  alert( 'SmartIT 학생 여러분 환영합니다!!' );
}
```

☞ 만약 parameter수가 여러 개 있다면 각 parameter를 ,(콤마)로 구분해준다. 이어서 function을 구성하는 코드의 모임인 '함수 본문(body)'을 { } 중괄호로 감싸 붙여준다.

```
function name(parameters) {
  ...funtion statement...
}
```

☞ 새롭게 정의한 함수는 함수 이름 옆에 ( ) 괄호를 붙여 호출할 수 있다. showMessage()같이 하면 된다.

```
function showMessage() {
  alert( 'SmartIT 학생 여러분 환영합니다!!' );
}

showMessage();
showMessage();
```

☑ JavaScript Function Expression (함수 표현식)

☞ JavaScript는 function을 특별한 종류의 값으로 취급한다. 다른 언어에서처럼 "특별한 동작을 하는 구조"로 취급되지 않는다.

☞ Function Declaration(함수 선언) 방식 외에 Function Expression(함수 표현식)을 사용해서 function을 만들 수 있다.

```
let sayHi = function() {
  alert( "Welcome to Semyung Univ." );
};
```

☞ function을 생성하고 variable에 값을 할당하는 것처럼 function가 variable에 할당되었다. function가 어떤 방식으로 만들어졌는지에 관계없이 function은 값이고, 따라서 variable에 할당할 수 있다. 위 예시에선 function이 variable sayHi에 저장된 값이 되었다.

☞ 위 예시를 간단한 말로 풀면 다음과 같다: "function을 만들고 그 function을 variable sayHi에 할당하기"

☞ function은 값이기 때문에 alert를 이용하여 function code(함수 코드)를 출력할 수도 있다.

```
function sayHi() {
  alert( "Welcome to Semyung Univ." );
}

alert( sayHi ); // function code가 보임
```

☞ 마지막 줄에서 sayHi 옆에 ( ) 괄호가 없기 때문에 function은 실행되지 않는다. 어떤 언어에선 ( ) 괄호 없이 function name만 언급해도 function이 실행된다. 그러나 JavaScript는 ( ) 괄호가 있어야만 function이 호출된다.

☞ JavaScript에서 function는 값이다. 따라서 function을 값처럼 취급할 수 있다. 위 코드에선 function code가 문자형으로 바뀌어 출력되었다.

☞ function는 sayHi()처럼 호출할 수 있다는 점 때문에 일반적인 값과는 조금 다르다. 특별한 종류의 값이다.

☞ 그러나 그 본질은 값이기 때문에 값에 할 수 있는 일을 function에도 할 수 있다.

☞ variable을 복사해 다른 변수에 할당하는 것처럼 function을 복사해 다른 변수에 할당할 수도 있다.

```
function sayHi() {                    // (1) function create
  alert( "Welcome to Semyung Univ." );
}

let func = sayHi;                     // (2) function copy
```

```
func();                                  // (3) 복사한 함수 실행(정상적으로 실행됨)
sayHi();                                 // 본래 function도 정상적으로 실행된다.
```

☞ function expression의 끝에 왜 세미 콜론 ;이 붙는지 의문이 들 수 있다. function declaration에는 세미 콜론이 없다.

```
function sayHi() {
  // ...
}

let sayHi = function() {
  // ...
};
```

이유는 if { ... }, for { }, function f { } 같이 중괄호로 만든 코드 블록 끝엔 ;이 없어도 된다. function expression은 let sayHi = ...; 과 같은 statement(구문) 안에서 값의 역할을 한다. 코드 블록{ }이 아니고 값처럼 취급되어 variable(변수)에 할당된다. 모든 statement의 끝엔 세미 콜론 ;을 붙이는 게 좋다. function expression에 쓰인 세미 콜론(;)은 function expression 때문에 붙여진 게 아니라, statement의 끝이기 때문에 붙여졌다.

☑ JavaScript Callback Function (콜백 함수)

☞ parameter가 3개 있는 함수, ask(question, yes, no)를 작성해보자. 각 parameter에 대한 설명은 다음과 같다.

✓ question => 질문

✓ yes => "Yes"라고 답한 경우 실행되는 function

✓ no => "No"라고 답한 경우 실행되는 function

☞ function은 반드시 question(질문)을 해야 하고, 사용자의 답변에 따라 yes() 나 no()를 호출한다.

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

function showOk() {
  alert( "You agreed." );
}

function showCancel() {
  alert( "You canceled the execution." );
}

// 사용법: function showOk와 showCancel가 ask function의 argument(인수)로 전달됨
ask("Do you agree?", showOk, showCancel);
```

☞ 이렇게 function를 작성하는 방법은 실무에서 아주 유용하게 쓰인다. 면대면으로 질문하는 것보다 위처럼 confirm 창을 띄워 질문을 던지고 답변을 받으면 간단하게 설문조사를 진행할 수 있다. 실제 상용 서비스에선 confirm 창을 좀 더 멋지게 꾸미는 등의 작업이 동반되긴 하지만, 일단 여기선 그게 중요한 포인트는 아니다.

☞ function ask의 argument(인수), showOk와 showCancel은 callback function 또는 callback 이라고 불린다.

☞ function를 function의 argument로 전달하고, 필요하다면 argument로 전달한 그 function를 "나중에 호출(called back)"하는 것이 cannback function의 개념이다. 위 예시에선 사용자가 "yes"라고 대답한 경우 showOk가 callback이 되고, "no"라고 대답한 경우 showCancel가 cakkback이 된다.

☞ 다음과 같이 function expression(함수 표현식)을 사용하면 코드 길이가 짧아진다.

```
function ask(question, yes, no) {
  if (confirm(question)) yes()
  else no();
}

ask(
```

```
  "Do you agree?",
  function() { alert("You agreed."); },
  function() { alert("You canceled the execution."); }
);
```

ask(...) 안에 function이 선언된 것이 보인다. 이렇게 이름 없이 선언한 함수는 anonymous function (익명 함수) 라고 부른다. anonymous function은 (변수에 할당된 게 아니기 때문에) ask 바깥에선 접근할 수 없다. 위 예시는 의도를 가지고 이렇게 구현하였기 때문에 바깥에서 접근할 수 없어도 문제가 없다.

JavaSCript를 사용하다 보면 callabck 활용한 코드를 아주 자연스레 만나게 된다. 이런 코드는 JavaScript의 정신을 대변한다.

JavaSCript에서 function은 "동작"을 나타내는 값이다. string(문자열)이나 number(숫자) 등의 일반적인 값들은 데이터를 나타낸다. function는 하나의 *동작(action)*을 나타낸다. 동작을 대변하는 값인 function을 variable 간에 전달하고, 동작이 필요할 때 이 값을 실행할 수 있다.

☑ Function Expression(함수 표현식) vs Function Declaration(함수 선언문)

Function Expression과 Function Declaration의 차이에 대해 알아보자다.

❖ 첫번째는 문법이다.

☞ Function Declaration: 함수는 주요 코드 흐름 중간에 독자적인 구문 형태로 존재한다.

```
// Function Declaration
function sum(a, b) {
  return a + b;
}
```

☞ Function Expression: 함수는 표현식이나 구문 구성(syntax construct) 내부에 생성된다. 다음 예시에선 함수가 할당 연산자 = 를 이용해 만든 "할당 표현식" 우측에 생성되었다.

```
// Function Expression
let sum = function(a, b) {
  return a + b;
};
```

❖ 두번째 차이는 JavaScript Engine이 언제 function를 생성하는지에 있다.

Function Expression은 실제 실행 흐름이 해당 함수에 도달했을 때 함수를 생성한다. 따라서 실행 흐름이 함수에 도달했을 때부터 해당 함수를 사용할 수 있다.
스크립트가 실행되고, 실행 흐름이 let sum = function…의 우측(Function Expression)에 도달 했을때 함수가 생성된다. 이때 이후부터 해당 함수를 사용(할당, 호출 등)할 수 있다.

그러나 Function Declaration은 조금 다르다.
Function Declaration은 Function Declaration이 정의되기 전에도 호출할 수 있다.
따라서 global Function Declaration은 스크립트 어디에 있느냐에 상관없이 어디에서든 사용할 수 있다.

이것이 가능한 이유는 JavaScript의 내부 알고리즘 때문이다. JavaScript는 스크립트를 실행하기 전, 준비단계에서 global(전역)에 선언된 Function Declaration을 찾고, 해당 함수를 생성한다. 스크립트가 진짜 실행되기 전 "초기화 단계"에서 함수 선언 방식으로 정의한 함수가 생성되는 것이다.
스크립트는 Function Declaration이 모두 처리된 이후에서야 실행된다. 따라서 스크립트 어디서든 Function Declaration으로 선언한 함수에 접근할 수 있는 것이다.

다음과 같은 예를 살펴본다.

```
sayHi("SmartIT");               // Hello, SmartIT

function sayHi(name) {
  alert( `Hello, ${name}` );
}
```

Function Declaration, sayHi는 스크립트 실행 준비 단계에서 생성되기 때문에, 스크립트 내 어디에서든 접근할 수 있다.

그러나 Function Expression으로 정의한 함수는 함수가 선언되기 전에 접근하는 게 불가능하다.

```
sayHi("SmartIT");               // error!

let sayHi = function(name) {    // (*) 마술은 일어나지 않는다.
  alert( `Hello, ${name}` );
};
```

Function Expression은 실행 흐름이 표현식에 다다랐을 때 만들어진다. 위 예시에선 (*)로 표시한 줄에 실행 흐름이 도달했을 때 함수가 만들어진다. 아주 늦다.

❖ 세 번째 차이점은, Scope 이다.

Stric Mode(엄격 모드)에서 Function Declaration이 코드 블록 내에 위치하면 해당 함수는 블록 내 어디서든 접근할 수 있다. 하지만 블록 밖에서는 함수에 접근하지 못한다.

Runtime에 그 값을 알 수 있는 변수 age가 있고, 이 변수의 값에 따라 함수 welcome()을 다르게 정의해야 하는 상황이다. 그리고 함수 welcome()은 나중에 사용해야 하는 상황이라고 가정해 본다.

Function Declaration을 사용하면 의도한 대로 코드가 동작하지 않는다.

```
let age = prompt("나이를 알려주세요.", 18);

// 조건에 따라 함수를 선언함
if (age < 18) {
  function welcome() {
    alert("안녕! 미성년자 입니다..");
  }
} else {
  function welcome() {
    alert("안녕하세요! 성인이네요!!");
  }
}

// 함수를 나중에 호출한다.
welcome();       // Error: welcome is not defined
```

Function Declaration은 함수가 선언된 코드 블록 안에서만 유효하기 때문에 이런 에러가 발생한다.

또 다른 예시를 살펴보자.

```
let age = 16;             // 16을 저장했다 가정한다.

if (age < 18) {
  welcome();               // \   (실행)
                           //  ┆
  function welcome() {     //  ┆
    alert("안녕!");        //  ┆  함수 선언문은 함수가 선언된 블록 내
  }                        //  ┆  어디에서든 유효하다
                           //  ┆
  welcome();               // /   (실행)

} else {
  function welcome() {
    alert("안녕하세요!");
  }
```

```
}

// 여기는 중괄호 밖이기 때문에
// 중괄호 안에서 선언한 Function Declaration은 호출할 수 없다.

welcome();       // Error: welcome is not defined
```

그럼 if 문 밖에서 welcome 함수를 호출할 방법은 없는 걸까?

Function Expression을 사용하면 가능하다. if문 밖에 선언한 변수 welcome에 Function Expression으로 만든 함수를 할당하면 된다.

이제 코드가 의도한 대로 동작한다.

```
let age = prompt("나이를 알려주세요.", 18);

let welcome;

if (age < 18) {
  welcome = function() {
    alert("안녕! 미성년자 입니다..");
  };
} else {
  welcome = function() {
    alert("안녕하세요! 성인이네요!!");
  };
}

welcome();        // 제대로 동작한다.
```

물음표 연산자 ?를 사용하면 위 코드를 좀 더 단순화할 수 있다.

```
let age = prompt("나이를 알려주세요.", 18);

let welcome = (age < 18) ?
  function() { alert("안녕! 미성년자 입니다.."); } :
  function() { alert("안녕하세요! 성인이네요!!"); };

welcome();       // 제대로 동작한다.
```

Function Declaration과 Function Expression 중 무엇을 선택해야 할까?
경험에 따르면 Function Declaration을 이용해 함수를 선언하는 걸 먼저 고려하는 게 좋다. Function Declaration으로 함수를 정의하면, 함수가 선언되기 전에 호출할 수 있어서 코드 구성을 좀 더 자유롭

게 할 수 있다.

Function Declaration을 사용하면 가독성도 좋아진다. 코드에서 let f = function(…) {…}보다 function f(…) {…} 을 찾는 게 더 쉽다. 함수 선언 방식이 더 “눈길을 사로 잡는다”.

그러나 어떤 이유로 함수 선언 방식이 적합하지 않거나, (위 예제와 같이) 조건에 따라 함수를 선언해야 한다면 Function Expression을 사용해야 한다

☑ JavaScript Array

key를 사용해 식별할 수 있는 value를 담은 collection은 Object라는 Data Structure를 이용해 저장하는데, object 만으로도 다양한 작업을 할 수 있다.
그러나 개발을 진행하다 보면 첫번째 element, 두번째 element, 세번째 element 등과 같이 order(순서)가 있는 collection이 필요할 때가 생긴다. 사용자나 물건, HTML element 목록같이 일목 요연하게 order를 만들어 정렬하기 위해서 이다.
order가 있는 collection을 다뤄야 할 때 obejct를 사용하면 order와 관련된 method가 없어 그다지 편리하지 않다. object는 태생이 order를 고려하지 않고 만들어진 Data Structure이기 때문에 object를 이용하면 새로운 property를 기존 property '사이에' 끼워 넣는 것도 불가능하다.
이럴 땐 order가 있는 collection을 저장할 때 쓰는 Data Structure인 Array를 사용할 수 있다.

❖ Declaration

방법 1)

```
let arr = new Array();
```

방법 2)

```
let arr = [];
```

☞ 대부분 두 번째 방법으로 배열을 선언하는데, 이때 [ ] 대괄호 안에 초기 element를 넣어주는 것도 가능하다.

```
let stds = ["SmartIt01", "SmartIT02", "SmartIT03"];
```

☞ 각 배열 요소엔 0부터 시작하는 숫자(index)가 매겨져 있다. 이 숫자들은 배열 내 순서를 나타낸다. 배열 내 특정 요소를 얻고 싶다면 [ ] 대괄호 안에 순서를 나타내는 숫자인 인덱스를 넣어주면 된다.

```
let stds = ["SmartIt01", "SmartIT02", "SmartIT03"];

alert( stds[0] );        // SmartIT01
alert( stds[1] );        // SmartIT02
alert( stds[2] );        // SmartIT03
```