

□ 개념 확인

(1) 괄호 안을 채워 넣으시오

- ① 자바 스크립트 객체는 키와 값으로 구성된 (프로퍼티)들의 집합이다
- ② 자바 스크립트 객체의 프로퍼티 값이 함수일 경우 일반 함수와 구분하기 위해 (메소드)라고 부른다
- ③ 자바 스크립트 객체의 프로퍼티 키는 빈 문자열을 포함하는 모든 (문자열)또는 심볼값을 사용한다
- ④ 프로퍼티 또는 메소드명 앞에 작성하는 (매개변수)는 생성자 함수가 생성할 인스턴스를 의미한다
- ⑤ 생성자 함수를 사용한 객체 생성시 (constructor) 키워드를 사용한다
- ⑥ 프로퍼티 값을 읽기 위해 대괄호 표기법을 사용할 경우 대괄호 내에 들어가는 프로퍼티 키는 반드시 (문자열) 이어야 한다
- ⑦ 생성자 함수 프로토타입을 사용할 경우 내부에는 (인스턴스 프로퍼티)만 존재한다.
- ⑧ 클래스에서 인스턴스 프로퍼티는 반드시 (생성자(constructor))에 정의되어야 한다
- ⑨ 객체 내에 특정 프로퍼티 존재 여부를 확인하려면 (in)연산자를 사용한다
- ⑩ (클래스)로 객체를 생성할 경우 반드시 new 연산자가 있어야 한다

(2) 리터럴 표기법으로 book 객체를 생성하는 문장을 선택하시오

- ① `let book={title:'js', price:30000}`
- ② `let book={title='js', price=3000}`
- ③ `let book={title='js'; price=3000}`
- ④ `let book=[title:'js', price:30000]`

1번

(3) 2번에서 생성된 book 객체에 접근하는 방법을 모두 선택하시오

- ① `book[title]`
- ② `book.title`
- ③ `book->title`
- ④ `book['title']`

2번, 4번

(4) 생성자 함수를 사용하여 객체를 정의하는 문장을 선택하시오

- ①

```
let Book = function(title, price){
  this.title=title;  this.price=price;
}
```

➔ 객체 리터럴

- ②

```
function Book(title, price){
  this.title=title;  this.price=price;
}
```

- ③

```
let Book = (title, price) => {
  this.title=title;    this.price=price;
}
```

➔ 람다식은 this 바인딩 불가능

- ④

```
function Book(title, price){
```

```
this.title=title; this.price=price;
}
```

```
Book.prototype.total=title;
```

→ title은 정의 되었지 않음

2번

(5) 4번의 생성자 함수를 사용하여 객체를 생성하는 문장을 제시하시오. 단, 매개값은 임의로 정할 것

==풀이==

```
let a = new Book("a", 20);
```

(6) 생성자 함수와 클래스로 객체를 생성하는 경우 차이점은 무엇인가?

==풀이==

생성자 함수는 옛날 방식이고 클래스는 신버전의 방식이다.

클래스에서는 constructor(){ } 함수로 클래스 내 생성자를 지정할 수 있고 클래스 내 모든 메서드는 프로토타입이다.

(7) 질문에 답하시오

① Object 생성자 함수를 사용하여 빈 객체를 생성하는 문장을 제시하시오. 단 객체명은 obj1

==풀이==

```
let obj1 = new Object();
```

② 1에서 생성된 객체에 다음과 같은 프로퍼티를 추가하고 임의의 값으로 초기화 한다.

time(자료타입 number), message(자료타입 string)

==풀이==

```
let obj1 = new Object();
obj1.time = 10;
obj1.message = "안녕하세요";
```

③ console.log(age in obj1); 실행 결과를 제시하시오.

==풀이==

```
false
> age in obj1
✖ ▶ Uncaught ReferenceError: age is not defined
  at <anonymous>:1:1
> 'age' in obj1
<< false
>
```

(8) 객체 생성과 메소드 호출을 참고하여 Book class를 작성하시오

```
const book = new Book('흑산', '김훈');
```

```
book.bwrite(); //객체 프로퍼티 값을 웹브라우저로 출력
```

==풀이==

```
class Book{
  constructor(a,b){
```

```

    this.a = a;
    this.b = b;
  }
  bwrite(){
    document.write(`a = ${this.a} b = ${this.b}`);
  }
}

```

- (9) 8에서 생성된 객체의 모든 프로퍼티를 순회하면서 출력하는 문장을 작성하시오. 힌트)for~in
==풀이==

```

for(let i in book){
  document.write(`${i} = ${book[i]}<br>`)
}

```

□ 개념 활용 응용 프로그래밍

- (1) 다음과 같은 속성과 메소드로 구성되는 객체를 제시된 방법으로 생성하고 결과를 확인하세요
- 속성 : 가수 이름, 곡명, 재생시간
 - 메소드 : play(cnt) - cnt 횟수만큼 반복 재생
 - 객체 생성 방법
 - 객체 리터럴

가수 : 이소라, 제목: 바람이 분다, 재생시간 : 3.5 => 1 번째 재생
 가수 : 이소라, 제목: 바람이 분다, 재생시간 : 7 => 2 번째 재생
 가수 : 이소라, 제목: 바람이 분다, 재생시간 : 10.5 => 3 번째 재생
 가수 : 이소라, 제목: 바람이 분다, 재생시간 : 14 => 4 번째 재생
 가수 : 이소라, 제목: 바람이 분다, 재생시간 : 17.5 => 5 번째 재생

[소스]

```

let play={
  name:"이소라",
  song:"바람이 분다",
  time: 3.5,
  play:function(cnt){
    for(let i = 1; i<=cnt; i++){
      document.write(`가수 : ${this.name}, 제목 : ${this.song}, 재생시간 :
${this.time*i} => ${i}번째 재생<br>`);
    }
  }
}
play.play(5);

```

[실행 결과]

가수 : 이소라, 제목 : 바람이 분다, 재생시간 : 3.5 => 1번째 재생
가수 : 이소라, 제목 : 바람이 분다, 재생시간 : 7 => 2번째 재생
가수 : 이소라, 제목 : 바람이 분다, 재생시간 : 10.5 => 3번째 재생
가수 : 이소라, 제목 : 바람이 분다, 재생시간 : 14 => 4번째 재생
가수 : 이소라, 제목 : 바람이 분다, 재생시간 : 17.5 => 5번째 재생

(2) 다음과 같은 속성과 메소드로 구성되는 객체를 생성하는 프로그램을 생성자 함수 프로토타입을 사용하여 구현한 후 제시된 결과처럼 동작할 수 있도록 프로그램을 작성하시오

- 속성 : 차량번호, 주행거리
- 메소드 : 주행거리를 dist 만큼 증가시키는 addMileage(dist) 메소드, 반환값 없음
차량번호와 주행거리를 문자열로 반환하는 toString()

127.0.0.1:5501 내용:

차량 번호와 주행거리를 입력하세요
더 이상 없으면 '완료'를 입력하세요

확인 취소

결과를 출력합니다

차량번호 : 50서1234 주행거리 : 150
차량번호 : 45머1345 주행거리 : 2000

힌트1) 데이터 입력은 prompt()함수를 사용하고 차량번호와 주행거리는 공백으로 구분한다

힌트2) 입력된 데이터는 split() 함수를 사용하여 구분한 후 객체 초기화에 사용한다

힌트3) 초기화된 객체는 Array에 저장한다.

[소스]

```
function car(a, b){
    this.number = a;
    this.dist = b;
}
car.prototype.addMileage = function(dist){
    this.dist += dist;
}
car.prototype.toString = function(){
    return `차량번호 : ${this.number}    주행거리 : ${this.dist}<br>`;
}
document.write(`<h1>결과를 출력합니다</h1><hr>`);

let arr = [];

while (true){
    let str = prompt("차량번호와 주행거리를 입력하세요\n더 이상 없으면 완료를 입력하세요");
    if(str == '완료'){
        break;
    }
    let sp = str.split(" ");
    arr.push(new car(sp[0], sp[1]));
}

for(let i = 0; i<arr.length; i++){
```

```
document.write(arr[i].toString());
}
```

[실행 결과]

결과를 출력합니다

차량번호 : 10하1028 주행거리 : 150
차량번호 : 48가1990 주행거리 : 700000
차량번호 : 24하2985 주행거리 : 0

127.0.0.1:5500 내용:

차량번호와 주행거리를 입력하세요
더 이상 없으면 완료를 입력하세요

(3) 다음과 같은 속성과 메소드로 구성되는 클래스 Account를 만들고 제시된 결과처럼 실행되는 프로그램을 작성하세요.

- 속성 : 예금주, 잔액
- 메소드
 - 매개변수로 받은 값 만큼 잔액을 증가하는 deposit(매개변수) 메소드, 반환값 없음
 - 매개변수로 받은 값 만큼 잔액을 감소하는 withdraw(매개변수) 메소드, 반환값 없으며 잔액이 적으면 "잔액부족" 출력

- 예금주와 잔액을 출력하는 display() 메소드, 매개변수 없음

```
현재 상태 입니다
예금주 : 스크립트
현재 잔액 : 50000

50000 예금 후 상태 입니다
예금주 : 스크립트
현재 잔액 : 100000

1000000을 인출하려고 합니다
잔액 부족 : 900000
```

[소스]

```
class Account{
    constructor(a,b){
        this.name = a;
        this.money = b;
        console.log("현재 상태 입니다.");
        this.display();
    }
    deposit(n){
        this.money+=n;
        console.log(n+"예금 후 상태입니다.");
        this.display();
    }
    withdraw(n){
        console.log(n+"을 인출하려고 합니다.");
        if(this.money - n < 0){
            console.log("잔액부족 : "+(n-this.money));
        }
    }
}
```

```

    }else{
        this.money-=n;
        console.log(n+"출금 후 상태입니다.");
        this.display();
    }
}
display(){
    console.log("예금주 : "+this.name);
    console.log("현재 잔액 : "+this.money);
    console.log("\n");
}
}

let kim = new Account('김지호', 50000);
kim.deposit(50000);
kim.withdraw(1000000);

```

[실행 결과]

현재 상태 입니다.
예금주 : 김지호
현재 잔액 : 50000
50000예금 후 상태입니다.
예금주 : 김지호
현재 잔액 : 100000
1000000을 인출하려고 합니다.
잔액부족 : 900000
> kim.deposit(100000);
100000예금 후 상태입니다.
예금주 : 김지호
현재 잔액 : 200000
< undefined
> kim.display();
예금주 : 김지호
현재 잔액 : 200000

- (4) 다음과 같은 속성과 동작을 갖는 대상을 자바스크립트 객체로 구현하고 테스트 하시오. 단, 클래스로 구현하고 테스트 결과는 console.log()를 사용하여 처리하시오.

백신종류 : 화이자, 연락처 : 010-2312-8723 접종현황: 미 접종
백신종류 : 화이자, 연락처 : 010-2312-8723 접종현황: 추가 1회
연락처 변경 후 출력
백신종류 : 화이자, 연락처 : 010-6543-7968 접종현황: 추가 1회

속성	값
백신	모더나, 화이자

접종 횟수	0
연락처	010-2193-5234
동작	내용
isFinished()	접종 횟수가 2이면 '접종 완료', 1이면 '추가 1회', 0이면 '미 접종' 반환
addShot()	접종 회수를 +1 증가, 만약 접종 회수가 2이면 증가 없음
changeTel(value)	연락처를 value값으로 변경

[소스]

```
class Vaccin{
  constructor(a,b){
    this.vaccin = a;
    this.count = 0;
    this.phone = b;
  }
  isFinished(){
    if(this.count==0){
      return "미 접종";
    }else if(this.count == 2){
      return "접종 완료";
    }else{
      return "추가 1 회";
    }
  }
  addShot(){
    if(this.count < 2){
      this.count++;
    }
  }
  changeTel(value){
    this.phone = value;
  }
}

let kim = new Vaccin('화이자', '010-2193-5234');
console.log(`백신 종류 : ${kim.vaccin}, 연락처 : ${kim.phone}, 접종현황 : ${kim.isFinished()}`);
kim.addShot()
console.log(`백신 종류 : ${kim.vaccin}, 연락처 : ${kim.phone}, 접종현황 : ${kim.isFinished()}`);
kim.changeTel('010-6543-7968')
console.log("연락처 변경 후 출력");
console.log(`백신 종류 : ${kim.vaccin}, 연락처 : ${kim.phone}, 접종현황 : ${kim.isFinished()}`);
kim.addShot()
console.log(`백신 종류 : ${kim.vaccin}, 연락처 : ${kim.phone}, 접종현황 : ${kim.isFinished()}`);
kim.addShot()
console.log(`백신 종류 : ${kim.vaccin}, 연락처 : ${kim.phone}, 접종현황 : ${kim.isFinished()}`);
```

[실행 결과]

백신 종류 : 화이자, 연락처 : 010-2193-5234, 접종현황 : 미 접종
백신 종류 : 화이자, 연락처 : 010-2193-5234, 접종현황 : 추가 1회
연락처 변경 후 출력
백신 종류 : 화이자, 연락처 : 010-6543-7968, 접종현황 : 추가 1회
백신 종류 : 화이자, 연락처 : 010-6543-7968, 접종현황 : 접종 완료
백신 종류 : 화이자, 연락처 : 010-6543-7968, 접종현황 : 접종 완료

>

(5) 2학년 조카의 구구단 학습 도우미 프로그램을 제시된 결과처럼 실행되도록 프로그램하세요.

- 1~9사이에 생성된 난수를 입력창에 제시된 결과처럼 출력하고, 답을 입력 받는다(10번 반복)

127.0.0.1:5500 내용:

1] 8*8 = ?

16

확인

취소

- 맞춘 회수에 10을 곱하여 점수를 계산한다.
- 계산된 점수가 90이상이면 '친구와 놀아도 됩니다', 80 이상이면 '한번 더 연습하세요', 70 이상이면 '두번 더 연습하세요', 70미만이면 '친구와 놀 수 없습니다'를 알림창으로 출력

127.0.0.1:5500 내용:

점수 : 60 -> 친구와 놀 수 없습니다

확인

- Gugudan 클래스를 정의하여 사용하도록 한다.

[소스]

```
class Gugudan{
    constructor(){
        this.sum = 0;
        this.play()
    }
    play(){
        let a;
        let b;
        for(let i = 0; i<10; i++){
            a = Math.floor(Math.random()*8)+2 //0~7 + 2 => 2~9
            b = Math.floor(Math.random()*9)+1 //0~8 + 1 => 1~9
            this.calc(i+1,a,b);
        }
        this.result(this.sum);
    }
    calc(i,x,y){
        let result = x * y;
        if(prompt(i+' ') '+x+"*" +y+" = ?') == result){
            this.sum+=10;
        }
    }
    result(a){
```



```

    if(a >= 90){ alert('점수 : '+ a + "-> 친구와 놀아도 됩니다."); }
    else if(a>=80){ alert('점수 : '+ a + "-> 한번 더 연습하세요."); }
    else if(a>=70){ alert('점수 : '+ a + "-> 두번 더 연습하세요."); }
    else{ alert('점수 : '+ a + "-> 친구와 놀 수 없습니다"); }
}
}

let gugu = new Gugudan();

```

[실행 결과]

The image displays four screenshots of a web application interface, each showing a message box with a score-based condition and a confirmation button.

- Top Left:** The message box displays "127.0.0.1:5500 내용:" followed by "점수 : 90-> 친구와 놀아도 됩니다." (Score: 90 -> You can play with friends). A green "확인" (Confirm) button is at the bottom.
- Top Right:** The message box displays "127.0.0.1:5500 내용:" followed by "점수 : 80-> 한번 더 연습하세요." (Score: 80 -> Practice once more). A green "확인" (Confirm) button is at the bottom.
- Bottom Left:** The message box displays "127.0.0.1:5500 내용:" followed by "점수 : 70-> 두번 더 연습하세요." (Score: 70 -> Practice twice more). A green "확인" (Confirm) button is at the bottom.
- Bottom Right:** The message box displays "127.0.0.1:5500 내용:" followed by "점수 : 0-> 친구와 놀 수 없습니다" (Score: 0 -> You cannot play with friends). A green "확인" (Confirm) button is at the bottom.

Below the other screenshots, there is a fifth screenshot showing a math problem interface:

- The message box displays "127.0.0.1:5500 내용:" followed by "3) 5*7 = ?".
- Below the text is an empty input field.
- At the bottom, there are two buttons: a green "확인" (Confirm) button and a white "취소" (Cancel) button.