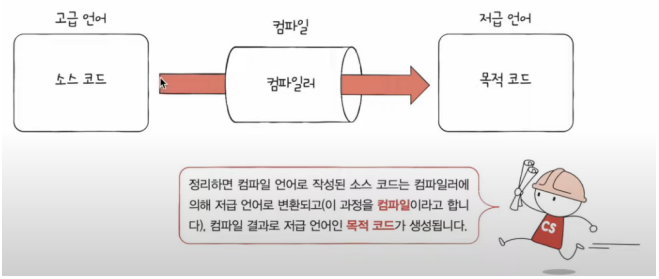
[11/02] Computing Language, Javascript install & Command Practice

- Computing Language : 컴퓨터에서 사용자가 원하는 결과를 만들어내기 위해 기계적, 프로그램적으로 작동되는 명령어가 집합된 code
- UI -> 프로그래밍 언어(컴퓨터인식 불가) -> 명령어 -> 컴퓨터 인식
    - 고급언어 -변환-> 저급언어
- 저급 언어(Low-Level programming language) : 컴퓨터가 이해하는 언어, 명령어
    - 기계어, 어셈블리어
- 고급 언어(High-Level programming language) : 컴퓨터가 이해하지 못하며 사람이 이해하고 작성하기 쉽게 만들어진 언어, 대부분의 프로그래밍언어가 여기에 속함
    - 기호 등을 사용하는 COBOL(사무), Pascal(수학,통계), Fortran(과학)
- 기계어(Machine Code) : 0과 1로 이루어진 명령어 비트 모음
    - 가독성을 위해 16진수로 표현하기도 함
- 어셈블리어(Assembly language) : 기계어를 읽기 편한 형태로 번역한 언어
    - 하드웨어와 밀접하게 맞닿아 있는 프로그램을 개발하는 임베디드 개발자, 게임, 정보보안 분야 등의 개발자가 주로 사용
    - 작성과 관찰의 언어로 프로그램이 어떤 과정으로 실행, 작동하는지 추적이 용이

- 컴파일언어(compile language) : 컴파일러에 의해 코드가 저급 언어로 변환되는 고급 언어
    - 컴파일(compile) : 컴파일 언어로 작성된 소스 코드를 저급 언어로 변환되는 과정
    - 컴파일러(compiler) : 컴파일을 수행해 주는 도구
        - 개발자가 작성한 소스 코드에 문법적 오류, 실행 가능, 불필요 코드 확인 수행하여 저급 언어로 컴파일함, 오류 발생시 컴파일에 실패함
        - 목적 코드(object code)[1] : 컴파일러를 통해 저급언어로 변환 성공된 코드
- 인터프리터언어(interpreter language) : 인터프리터에 의해 코드가 한 줄씩 실행되는 고급 언어
    - ex) python
    - 인터프리터(interpreter) : 소스 코드를 한 줄씩 저급 언어로 변환, 실행해주는 도구
        - 컴파일러와는 달리 문법 오류되기전까지의 수행은 가능
        - 컴파일러가 언어를 한 줄씩 해석& 실행하므로 컴파일러에 비해 느린 편
- 컴파일언어와 인터프리터언어의 구분이 모호하기도 하며, python도 컴파일을 수행함
    - 둘다 가능하며 각각의 방식이 존재함



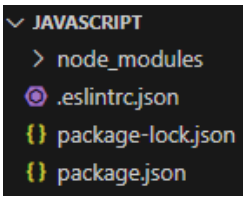
[1] 심화 : 목적 코드는 실행파일이 되기 위해 링킹(linking) 작업을 통해 더하기, 출력 명령어로 실행된다.

- 컴퓨터 언어의 구조
  - 구조적 언어 : 어려운 문법, 시스템 프로그램 개발 ex) C, COBOL, Pascal, Fortran
  - 객체지향[2] 언어 : 상속, 추상화, 다형성, 캡슐화 ex) C++, Java, Python
  - 스크립트 언어 : 컴파일이 아닌 인터프리터 방식 ex) C#, Javascript, ASP, JSP, PHP

- 객체(object) : 모든 실재하는 대상
- 객체 지향 프로그래밍 : 객체를 추상화 시켜 속성(state)와 기능(behavior)으로 분류한 후 이것을 다시 변수(varialble)과 함수(function)로 정의
  - 추상화(Abstraction) : 프로그램에 필요한 부분만 파악하여 추출
  - 상속(Inheritance) : 부모와 자식의 관계처럼 상위 클래스의 내용을 하위가 받음
  - 다형성(Polymorphism) : 하나의 객체가 다른 여러 객체로 재구성
  - 캡슐화(Encapsulation) : 데이터와 알고리즘을 하나로 묶어 외부에서는 확인불가

- Javascript 초기 설치 & 설정
  - VS Code : File > Close folder 상태로 시작
  - npm init -yes : 패키지 JSON 생성
  - npm install -g eslint : eslint 설치
    - g : 외부에서 해당하는 값 호출, 하위 디렉토리에 대한 선언
  - npx eslint - - init : 환경설정 초기화 (=npm init @eslint/config)
  - To check syntax and find problems > JavaScrip modules (import/export) > None of these > No > Node > JSON > YES > NPM
    - 선택 안될경우 spacebar 로 지정/해제

- Javascript 설치 완료시



파일 생성 및 추가 설정 불필요

- 실행 (node 파일명)
  - ex) node exm01.js

- package.json

```
{} package.json > ...
{
  "name": "javascript",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test
          specified\" && exit 1"  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "eslint": "^8.52.0"
  }
}
```

  - "main" : "index.js" == 프로그램 실행시 기준 파일
  - ▷Debug : 오류시 출력할 메시지 설정
  - "keywords" : 안내 태그를 붙일 곳
  - "devDependencies" : 라이브러리(작업을 도와주는 툴 집합) 목록

---

[2] 객체 지향 프로그래밍의 4가지 특징 :추상화, 상속, 다형성, 캡슐화

- 자바스크립트와 자바의 차이 : 편리성

```
let s="hello world";
for (let i=0; i<s.length; ++i)
{let ch= s[i];
console.log(ch);}
// 자바스크립트의 for 문
```

```
public class for{
    public static void main(string[] args){
        string s="hello world"
        for (int i=0; i<s.length(); ++i){
            char ch = s.charAt(i);
            system.out.println(ch);
        }
    }
}
// 자바의 for 문
```

- 자바스크립트의 특징 : let

```
a=1;
b=3;
console.log("a="+a);
console.log("b="+b);
console.log(a+b);
```

```
a=1
b=3
4
```

  - a : 변수명 / 1 : 변수값
  - let : 상수로 지정 ex) let a=1;
    - 후위우선 : 자바는 상수로 지정해도 후술하는 변수에 대한 설정이 우선
    - 자료형과 자료형이 아닌 것에 대한 지정이 없기 때문에 다소 편리
      - 정수형인 int 나 문자형인 char 대신 let으로 통일

- for 문과 console.log() 에서는 ; 생략 #뭐야..세미콜론 찍던가 말던가 하나만 하지

```
for (let i = 0; i<10; i++)
console.log(i);
```

```
0
1
2
3
4
5
6
7
8
9
```

  - for : 반복문 / let i = 0 : 시작값 / i < 10 : 종료 / i++ : 조건문(증가값)
  - 전위연산자 : ++i , 내부 타입의 경우 속도는 같으나 객체의 경우 전위연산자가 빠름
    - 후위연산자는 현재 값을 리턴후 값을 증가하는 복사생성 방식

- 비교연산자

연산자	개요	사용 예	결과
==	좌변과 우변의 값이 같을 경우 true	3 == 3	true
!=	!= 좌변과 우변의 값이 같지 않을 경우, true	3 != 3	false
<	좌변이 우변보다 작은 경우, true	3 < 7	true
<=	좌변이 우변보다 작거나 같을 경우, true	3 <= 3	true
>	좌변이 우변보다 클 경우, true	3 > 7	false
>=	좌변이 우벼보다 크거나 같을 경우, true	3 >= 3	true
===	좌변과 우변의 값이 같고 데이터 형도 같은 경우, true	3 === 3	true
!==	좌변과 우변 값이 같지 않거나 데이터 형이 다른 경우, true	3 !== 3	false
?:	조건 식? 식1:식2, true 인 경우 식1을 false 인 경우 식2를 수행	<x==1) ? 1:0	1 또는 0

  - 왼쪽 정렬 : a>b 와 b<a의 결과는 같으나 항상 큰 쪽을 왼쪽으로 a>b 정렬 권장

- for 문과 문자열 [ ] 지정의 활용

```
let s="hello world";
for (let i=0; i<s.length; ++i)
{let ch= s[i];
console.log(ch);}
```

```
h
e
l
l
o

w
o
r
l
d
```

  - length : 전체 길이의 갯수
  - s.length : s의 길이만큼
  - s[i] : s의 [i] 번째
    - 여기서 i는 for문의 ++i 로 인해 0~12를 의미
    - s[1] = h / s[2] = e / s[3] = l / s[4] = l

- 조건문
  - 반복문 : for, while(조건 검사후 실행), do-while(우선 실행후 반복여부 결정)

```
let i = 0;
for (i=0; i<10; ++i) {
    console.log(i);
}
```

```
let i = 0;
while (i<10) {
    console.log(i);
    ++i;
}
```

    - 결과는 동일한 for문과 while 문

```
let j=0;
while (j<10) {
    console.log(j);
    ++j;
}
```

```
let k=0;
do{
    console.log(k);
    ++k;
} while(k<10);
```

    - 결과는 동일한 while 과 do while 문

  - if 문의 continue 와 break

```
let i = 0;
for(i=0; i<100; ++i){
    if(i%2 ==1) continue;
    console.log(i);
    if(i>=20) break;
}
```

```
0
2
4
6
8
10
12
14
16
18
20
```

    - if(i%2 ==1) continue : i를 2로 나누고 1이면 continue(넘어가고) 아니면 출력
    - if(i>=20) break : i 가 20보다 크거나 같으면 break(중지)
    - 자바스크립트 처리상 let i=0; 이 없어도 결과는 같음

- 지역변수 및 전역변수

```
let s1="papa";
const s2="papa";
console.log(s1,s2);
```
```
papa papa
```

  - 지역변수(let) : let 은 해당 공간내에서의 작동
  - 전역변수(const) : const는 전체 공간에서의 작동

- 0 이상 55이하의 모든 정수를 더하는 코드를 반복문으로 구현

```
let a=0;
for (i=1; i<=55; ++i) {
    a=i+a;
}
console.log(a);

let sum=0;
let i=1;
while(i<=55){
    sum=sum+i;
    i++;
}
console.log(sum);
```
```
a=i+a
0=1+0
1=2+1
3=3+3
6=4+6
10=5+10
.
.
.
1485=55+1485
1540
```

  - let i=0; 가 생략되어도 자바스크립트는 용인
  - 그러나 **for** 문의 계산식 **a** 에 대한 정의는 필요
  - a+= i

- 1 이상 100이하의 모든 3의 배수를 더해서 화면에 출력하는 코드를 반복문으로 구현

```
let a=0;
for (i=1; i<=100; ++i) {
    if(i%3==0)
    a+=i
}
console.log(a);
```
```
1683
```

  - if(i%3==1) continue;
  - if(i%3==2) continue;

- 1 이상 100이하의 모든 7의 배수와 16의 배수를 더해서 화면에 출력하는 코드를 반복문으로 구현

```
let a=0;
let b=0;
for (i=1; i<=100; ++i) {
    if(i%7==0)a+=i;
    if(i%16==0)b+=i;
}
console.log(a+b);
```
```
1071
```

  - let i=0; 선언시 화면상 붉은 줄이 사라짐

- 자료형
  - 숫자형(num) / 자료형(string)

```
let a=3, b="3";
console.log(a,b)
```

  - True / False

```
let a=3, b="3";
console.log(a,b);
console.log(a==b);
console.log(a!=b);
```

```
3 3
true
false
```

    - 약타입언어(weak-type) : 자바스크립트는 자료형을 엄격히 구분하지 않음

```
let a=3, b="3";
console.log(a,b);
console.log(a===b);
console.log(a!==b);
```

```
3 3
false
true
```

    - 강타입언어(strong-type) : 확실한 구분을 위해

  - 자료형 확인 : typeof

```
let a=3, b="3", c=true, d;
console.log(typeof a);
console.log(typeof b);
console.log(typeof c);
console.log(typeof d);
```

```
number
string
boolean
undefined
```

    - undefined : 자료의 공간이 없는 상태
    - null : 공간은 있으나 값은 비어있는 상태
    - 단, 자바스크립트는 둘다 상관하지 않는 편

  - 자료형의 지정 : toString();

```
let a=3.141592;
console.log(a);
console.log(typeof a);
```

```
3.141592
number
```

```
let s=a.toString();
console.log(s);
console.log(typeof s);
```

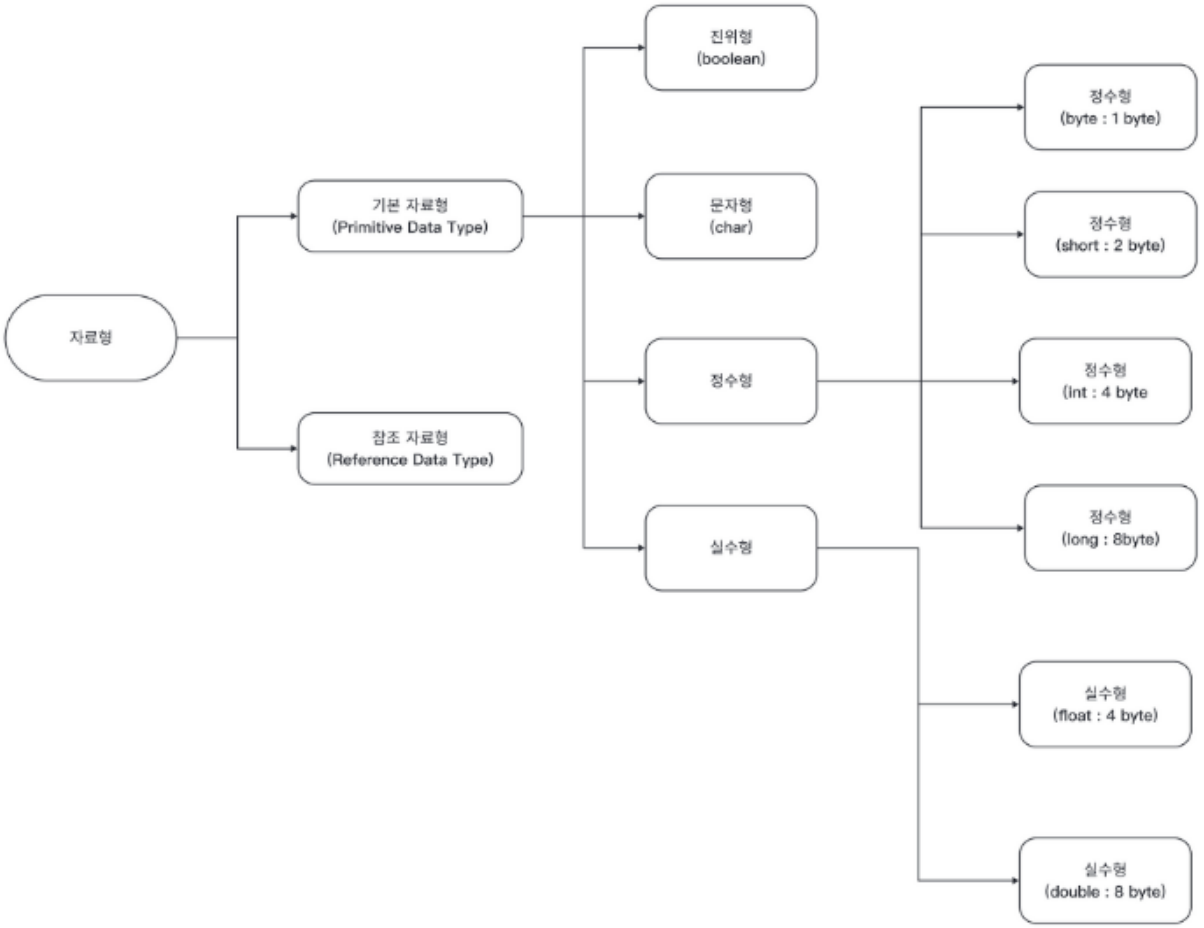
```
3.141592
string
```

```
let a=3.141592;
let s=a.toString();
let n=Number(s);
console.log(n);
console.log(typeof n);
```

이중변형도 적용

- 자바의 자료형(Data Type)
  - 기본자료(Primitive Type)
    - 수치(number) / 문자(string) / 참거짓(boolean) / 미정의(undefined)
  - 복잡자료(complex type)
    - 기능(function) : 실행문장 / 객체(object) : 자료 또는 처리 방법의 모음
  - 참조형자료(Reference Type)

- [Java] 자료형(Data Type) 이해하기 : 기본 / 참조 자료형, 래퍼 클래스



분류	타입	크기	기본값	범위
정수형	byte	1byte	0	-128 ~ 127
정수형	short	2byte	0	-32768 ~ 32767
정수형	int	4byte	0	-2147483648 ~ 2147483647
정수형	long	8byte	0L	- 9223372036854775808 ~ 9223372036854775807
문자형	char	4byte	'\u0000'	0 ~ 65535 (\u0000 ~ \uffff)
실수형	float	4byte	0.0f	single-precision 64-bit IEEE 754 floating point
실수형	double	8byte	0.0d	double-precision 64-bit IEEE 754 floating point
논리형	boolean	1byte	false	true, false

- random 함수

```
for (let i=0; i<5; i++){
    let a= Math.floor(Math.random()*10)
    console.log(a);
}
```

```
0
4
7
9
0
```

  - random : 0~1 사이의 소수점까지 등장 / let 에 의해 int 가 등장하는 편
  - i<5 : 5개가 순서대로 출력됨
  - *10 : 10미만으로 최대값 출력 # 0~10 등장
    - *배수 : *40은 40배수가 등장?!
  - *3+1 : +1은 1이상으로 최소값 출력 #1,2,3 만 등장

```
let b=2;
for (let i=0; i<1; i++){
    let a= Math.floor(Math.random()*3+1)
    console.log(a);
    console.log(a == b);
}
```

```
2     3      1
true  false  false
```

    - 임의의 비밀번호, 가위바위보 등에 적용

- 10 이상 20 이하의 정수를 임의로 30개 출력하는 코드를 구현

```
for (let i=0; i<30; i++){
    let b= Math.floor(Math.random()*11+10)
    console.log(b);
}
```