**ㅇ 컴퓨터란?**

**컴퓨터란, 입력 받은 데이터를 정해진 규칙대로 처리해서 처리 결과를 저장하고 출력하는 전자 장치이다.**

* **‘입력 받은 데이터를’**
  + 컴퓨터는 사람으로부터 데이터를 받아들입니다.
  + 즉, 컴퓨터는 데이터를 **입력장치**를 통해 입력받습니다.
* **‘정해진 규칙대로’**
  + 정해진 규칙은 프로그램을 의미합니다.
  + 프로그램은 소프트웨어 및 애플리케이션이라고도 합니다.
* **‘처리해서’**
  + 위에서 언급한 정해진 규칙대로 연산을 수행합니다.
  + 즉, 컴퓨터는 **중앙처리장치**를 통해 연산을 수행합니다.
* **‘처리 결과를 저장하고’**
  + 컴퓨터는 데이터를 저장할 수 있는 장치를 가지고 있습니다.
  + 즉, 컴퓨터는 처리 결과물을 **기억장치**에 저장합니다.
* **‘출력하는 전자 장치이다’**
  + 컴퓨터는 데이터 처리 결과를 사람이 감각할 수 있는 형태의 정보로 사람에게 전달해줍니다.
  + 즉, 컴퓨터는 처리 결과물을 **출력장치**를 통해 출력합니다.

ㅇ 컴퓨터의 구성요소

* **‘정해진 규칙대로’ → 소프트웨어**
* **‘입력 받은 데이터를 처리해서 처리 결과를 저장하고 출력하는 장치이다’ → 하드웨어**

ㅇ 시스템 소프트웨어

 **응용 소프트웨어의 실행 환경이다.**

* Ex. Mac, Windows, Linux와 같은 운영체제 및 Node.js, JRE와 같은 런타임 환경

 **하드웨어 장치를 관리한다.**

* Ex. 키보드 드라이버, 마우스 드라이버, USB 드라이버와 같은 각종 장치 드라이버들

ㅇ 응용소프트웨어

응용 소프트웨어는 소프트웨어에서 시스템 소프트웨어를 제외한 모든 소프트웨어를 의미합니다. 줄여서 애플리케이션 및 앱이라고도 하며, 일반적으로 프로그램이라고도 불립니다. 종류는 사무용, 교육용, 멀티미디어, 통신용, 게임 등 매우 다양합니다.

ㅇ 하드웨어

하드웨어는 컴퓨터를 구성하는 기계적 장치들의 집합입니다. 사람으로 비유하면 신체 그 자체라고 할 수 있겠습니다. 즉, 여러분들이 손으로 만질 수 있는 컴퓨터의 물리적인 장치들을 하드웨어라고 합니다. 컴퓨터의 정의에서 살펴본 것처럼, 하드웨어는 중앙처리장치, 기억장치, 입력장치, 출력장치로 분류됩니다. 그리고, 각 장치는 시스템 버스로 연결되어져 있습니다.

ㅇ 중앙처리장치 ( CPU )

CPU는 프로그램의 실행과 데이터 처리를 담당하는 컴퓨터의 핵심 장치로, 프로세서라고도 합니다. CPU는 제어장치, 연산장치, 레지스터로 구성되는데, 각 요소의 동작이 어우러져 CPU의 기능을 수행합니다.

* **제어장치**
  + CPU의 모든 동작과 데이터 흐름을 통제하고 관리합니다.
  + 운영체제와 소통하며 컴퓨터의 자원을 관리합니다.
* **연산장치**
  + 프로그램에 기술되어져 있는 명령어를 수행합니다.
  + 코드를 실질적으로 실행시키는 장치입니다.
* **레지스터**
  + CPU 내에 존재하는 작고 빠른 기억장치입니다.
  + 연산장치의 동작에 필요한 데이터들을 임시적으로 보관하며 제공합니다.

**기억장치 (Memory Device)**

기억장치는 저장장치라고도 하며, CPU의 동작에 필요한 데이터들을 장기적으로 또는 단기적으로 보관합니다. 여기에서 장기적으로 저장하는 장치를 **보조기억장치**라고 하며, 단기적으로 데이터를 저장하는 장치를 **주기억장치**라고 합니다.

* **보조기억장치**
  + 컴퓨터에게 필요한 모든 정보를 반영구적으로 저장합니다.
  + 큰 용량의 데이터를 저장할 수 있지만, 데이터를 읽고 쓰는 속도가 느립니다.
  + CPU의 연산 속도는 엄청나게 빠른 반면, 보조기억장치는 느리기 때문에 CPU는 보조기억장치와 직접적으로 소통하지 않습니다.
  + Ex. SSD(Solid State Drive), HDD (Hard Disk Drive)
* **주기억장치**
  + 컴퓨터가 프로그램을 수행하는 동안 필요한 모든 데이터들을 저장합니다.
  + 큰 용량의 데이터를 저장할 수 없지만, 데이터를 읽고 쓰는 속도가 빠릅니다.
  + CPU의 연산 속도를 어느 정도 따라 잡을 수 있을 정도로 빠르기 때문에 CPU는 주기억장치와 직접적으로 소통합니다.
  + Ex. RAM(Random Access Memory), ROM (Read Only Memory)

ㅇ 기억장치의 계층구조



### ㅇ 시스템 버스 (System Bus)

시스템 버스는 CPU를 포함한 각 하드웨어 장치 간의 통로입니다. 즉, 각 하드웨어 장치를 물리적으로 연결한 전선을 의미하기 때문에 시스템 버스 또한 하드웨어에 포함됩니다. 시스템 버스는 다음과 같이 이루어져 있습니다.

* **데이터 버스**
  + 하드웨어 간에 데이터를 전달하는 통로입니다.
* **주소 버스**
  + 데이터가 어디에 도착해야 하는지에 대한 정보를 전달하는 통로입니다.
* **제어 버스**
  + CPU의 제어장치가 생성한 제어 신호를 다른 장치로 전달하는 통로입니다.

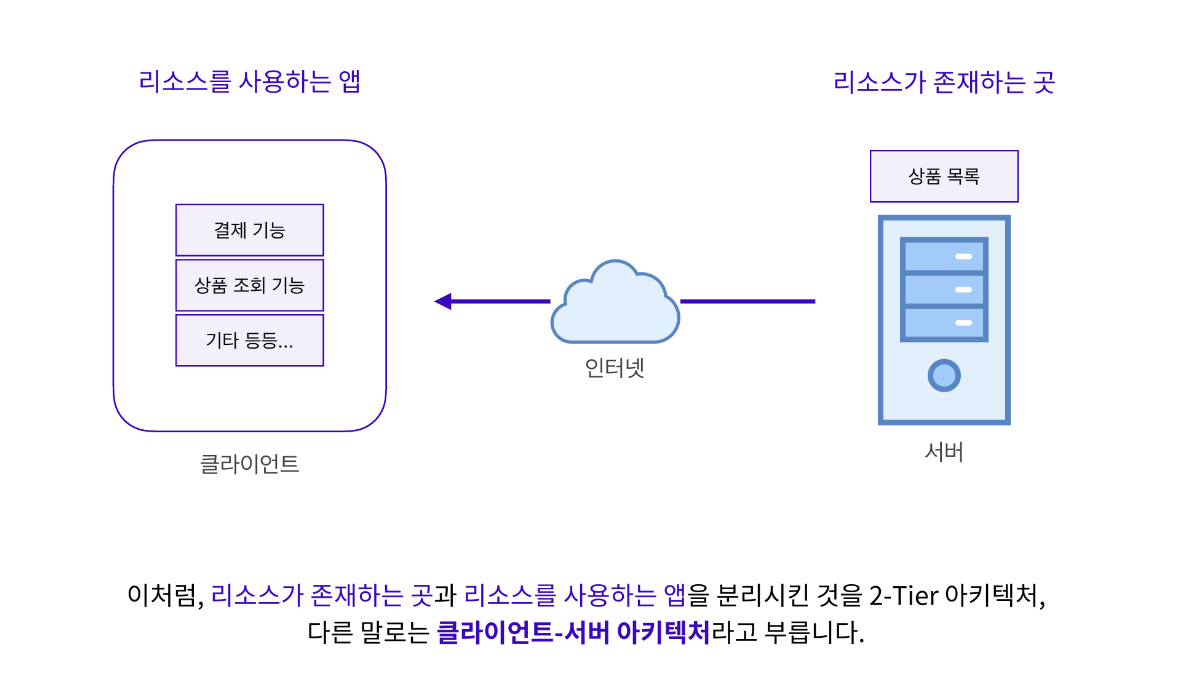
## 컴파일

프로그래밍 언어는 그저 사람의 언어와 컴퓨터의 언어를 매개할 뿐, 컴퓨터는 프로그래밍 언어로 작성된 소스 코드를 이해하지 못합니다. 그래서 소스 코드를 작성한 후에 컴퓨터에게 내용을 전달하려면 소스 코드를 기계어 코드로 변환시켜주어야 합니다. 이처럼 프로그래밍 언어로 작성된 소스 코드를 컴퓨터가 이해할 수 있는 기계어 코드로 번역하는 과정을 **컴파일(Compile)**이라고 합니다. 컴파일이 완료된 코드는 이제 컴퓨터가 이해할 수 있습니다.

## 빌드

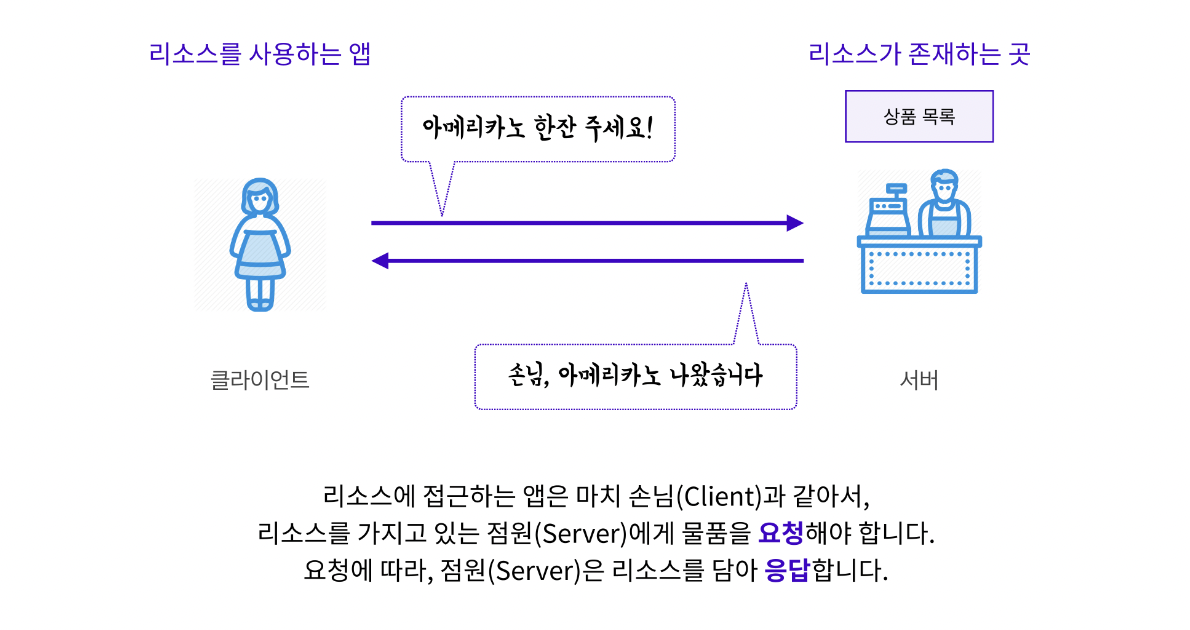
앞서 언급한 컴파일은 **빌드(Build)**라는 과정의 일부입니다. 빌드는 작성한 소스 코드를 실행할 수 있는 산출물로 변환시키는 과정을 의미하는데, 그 과정 중에 소스 코드를 기계어 코드로 번역하는 컴파일 과정에 포함되어 있습니다. 컴파일을 마친 코드는 기계어 코드로 번역되어 컴퓨터가 이해할 수 있을 뿐, 아직 사용자가 소스 코드를 실행할 수는 없습니다. 프로그램이 실행되려면 .exe또는 .msi 확장자를 가진 실행 파일로 만들어주는 빌드 과정이 필요합니다. 빌드가 완료되어야 소스 코드가 비로소 실행시킬 수 있는 하나의 프로그램으로 완성됩니다.

ㅇ 클라이언트-서버 아키텍처



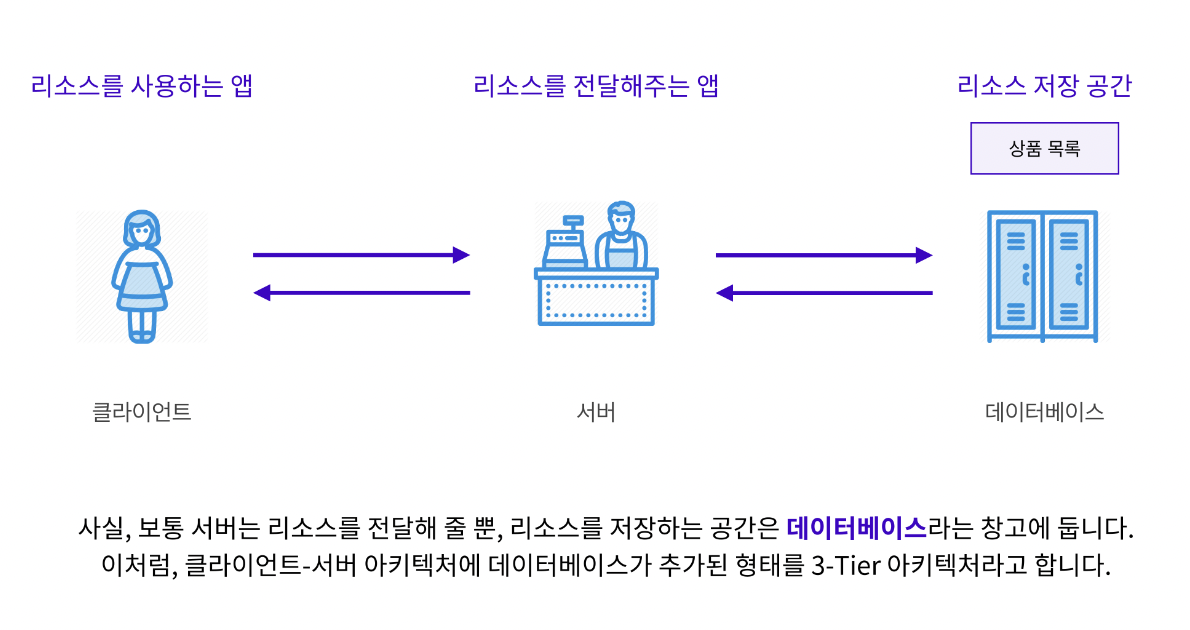
이렇게 상품 정보 같은 리소스가 존재하는 곳과 리소스를 사용하는 앱을 분리시킨 것을 2티어 아키텍처, 또는 클라이언트-서버 아키텍처라고 부릅니다. 리소스를 사용하는 앱이 바로 "클라이언트", 리소스를 제공(serve)하는 곳은 "서버"라고 부릅니다. 그림을 잘 보세요. 상품 정보는 서버에서 다루고 있습니다. 클라이언트는 단지 상품 정보를 조회할 뿐입니다.

ㅇ 클라이언트-서버 아키텍처



클라이언트와 서버는 요청과 응답을 주고받는 관계입니다. 클라이언트-서버 아키텍처에서는 요청이 선행되고 그 후에 응답이 옵니다. 요청하지도 않았는데 응답이 오는 경우는 없습니다.

**ㅇ 3-Tier 아키텍처**



일반적으로 서버는 리소스를 전달해 주는 역할만 담당합니다. 리소스를 저장하는 공간을 별도로 마련해 두는데 이 공간을 "데이터베이스"라고 부릅니다. 데이터베이스는 창고와 같은 역할을 합니다. 이처럼 기존 2티어 아키텍처에 데이터베이스가 추가된 형태를 3티어 아키텍처라고 부릅니다.

ㅇ 벡엔드란

사용자 눈에 보이지 않지만, 상품 정보를 API로 노출한다든지, 로그인/로그아웃, 권한 관리 등의 사용자 인증을 주로 다루는 개발자는 백엔드 개발자라고 부릅니다. (서버가 단순히 전달만 하는 게 아닙니다!) 백엔드 개발자는 데이터베이스 등의 시스템 설계까지 도맡아서 하는 경우도 많습니다.

ㅇ HTML

- HTML / CSS / Javascript 개념 설명

1) HTML :

- 웹 문서를 구조적으로 표현하기 위한 언어

2) CSS:

- 구조적인 문서를 어떻게 시각적으로 표현하는지에 대한 언어

3) JavaScript:

- 웹 문서를 표현하기위한 스크립팅언어이자, Node.js 를 활용해 서버제작 가능

ㅇ 각 태그 설명

1) HTML : 시작 태그로, 문서 전체의 틀을 구성

2) head : 문서의 메타데이터를 선언, 문서의 제목, 브라우저의 탭에 보여진다.

3) body : 문서의 내용을 담는 곳

ㅇ 가장 많이 사용되는 태그들

1) div : division ( 한 줄을 차지 )

2) span : span ( 컨텐츠 크기만큼 차지 )

3) img : Image

4) a : link

5) ul & li : 정렬되지 않은 리스트와 요소들

6) input : Text, Radio, Checkbox

- 형태 : <input type = "text" placeholder="type here" />

7) textarea : Multi-line Text Input

8) button : butto

9) main : body의 주요 콘텐츠를 나타낸다.

ㅇ CSS 의 Id / class

- Id

1) #으로 선택

2) 한 문서에 단 하나의 요소에만 적용

3) 특정 요소에 이름을 붙이는데 사용

- class

1) .으로 선택

2) 동일한 값을 갖는 요소 많음

3) 스타일의 분류에 사용

ㅇ display:flex 옵션

: 기본적으로 div는 위에서 아래로 수직으로 정렬되며, 가로로 넓은 공간을 차지한다.

이 옵션을 통해 가로로 공간을 가지게 할 수 있다.

- 부모 박스 요소에 적용

- 자식 박스의 방향과 크기를 결정하는 레이아웃 구성 방법이다.

ㅇ flex-direction

- row : 가로 정렬

- column : 세로 정렬

- row-reverse : 가로 - 오른쪽에서 왼쪽으로 정렬

- column-reverse : 세로 - 밑에서 위로 정렬

ㅇ flex-wrap

: 하위 요소의 크기가 상위 요소의 크기를 넘으면 자동 줄바꿈

- nowrap : 기본값

- wrap : 위에서부터 왼쪽에서 오른쪽으로 줄바꿈

- wrap-reverse : 밑에서부터 왼쪽에서 오른쪽으로 줄바꿈

ㅇ justify-content

: 축 수평 정렬 방향을 정해준다.

- row 는 default로 가로방향이고, column : 세로방향

- flex-start, end, center, space-between, space-around

ㅇ align-items

: 축 수직 방향 정렬을 한다.

요소들이 가로로 정렬되있으면 세로 방향으론 어떻게 정렬할지, 세로로 정렬이라면 가로방향으로 어떻게 정렬할 것인지를 정하는 속성이다.

## flex 속성의 값

flex 속성에는 세 가지 값을 지정해줄 수 있습니다. 각 값이 의미하는 것은 다음과 같습니다.

1

flex: <grow(팽창 지수)> <shrink(수축 지수)> <basis(기본 크기)>

grow(팽창 지수) 는 요소의 크기가 늘어나야 할 때 얼마나 늘어날 것인지, shrink(수축 지수) 는 요소의 크기가 줄어들어야 할 때 얼마나 줄어들 것인지, basis(기본 크기) 는 늘어나고 줄어드는 것과 상관없이 요소의 기본 크기는 얼마인지를 의미합니다. 각 값이 어떻게 작용하는지는 잠시 후 살펴보겠습니다.

자식 요소에 flex 속성을 따로 설정해주지 않으면 다음과 같은 **기본값**이 적용되며, 왼쪽에서부터 오른쪽으로 콘텐츠의 크기만큼 배치됩니다.

1

flex: 0 1 auto;

이 순서와 기본값은 반드시 기억해주세요. **flex: grow shrink basis, flex: 0 1 auto**

꼭 flex 속성 안에 세 가지 값을 한 번에 설정해줄 필요 없이, 다음과 같이 각 값을 따로 지정해줄 수 있습니다.

1

2

3

flex-grow: 0;

flex-shrink: 1;

flex-basis: auto;

### 1. grow : 자식 박스는 얼마나 늘어날 수 있을까요?

grow(팽창 지수) 는 요소의 크기가 늘어나야 할 때 **얼마나 늘어날 것인지**를 의미한다고 했습니다. 어떤 의미인지 알아보기 위해서 챕터 2-1에서 사용했던 HTML에서 자식 <div> 요소들에 id를 추가해서 grow 값을 변경하며 화면 변화를 확인해보겠습니다.

1

2

3

4

5

<main>

<div id="box1" class="box">box1</div>

<div id="box2" class="box">box2</div>

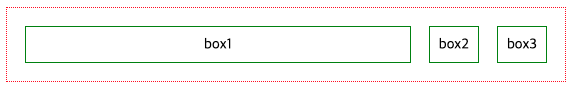
<div id="box3" class="box">box3</div>

</main>

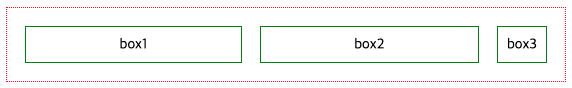
flex 속성을 설정하기 전의 모습입니다. grow 의 기본 값인 0은 빈 공간이 있어도 늘어나지 않음을 의미합니다. 따라서 빈 공간이 있음에도 박스들이 늘어나지 않습니다. 그럼 grow 의 값을 변경하며 어떤 변화가 생기는지 확인해봅시다.



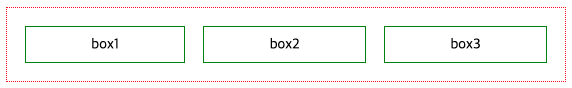
우선 box1만 flex-grow : 1 로 설정해보겠습니다. flex-grow 의 기본값은 0이므로 나머지 박스는 flex-grow : 0 인 상태입니다.



이번에는 box2도 flex-grow : 1 로 설정해보겠습니다. box3만 flex-grow : 0 인 상태입니다.



box3도 flex-grow : 1 로 설정해보겠습니다. 세 박스 모두 flex-grow : 1 인 상태입니다.



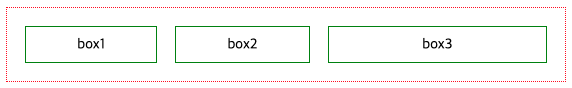
어떤 느낌인지 감이 오시나요? grow(팽창 지수) 는 정렬축 방향으로 빈 공간이 있을 때, 각 자식 요소들이 얼마나 늘어나서 남는 공간을 차지할 것인지 비율을 정하는 것이라고 생각하면 이해하기 쉽습니다.

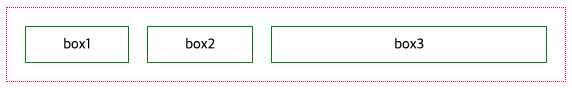
즉, 첫 번째 예시에서 grow는 box1 : box2 : box3 = 1 : 0 : 0 이므로 box1이 모든 공간을 차지하고,

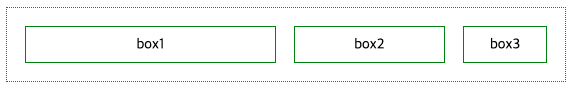
두 번째 예시에서 grow는 box1 : box2 : box3 = 1 : 1 : 0 이므로 box1과 box2가 1:1로 공간을 나눠 가지고,

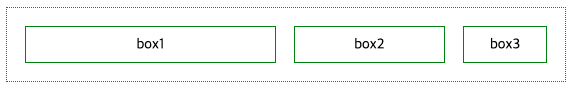
세 번째 예시에서 grow는 box1 : box2 : box3 = 1 : 1 : 1 이므로 세 박스가 1:1:1로 공간을 나눠 가진 것입니다.

예시를 좀 더 살펴보겠습니다.

[예시 1] 팽창지수가 1 : 1 : 2 일 때

[예시 2] 팽창지수가 1 : 1 : 4 일 때

[예시 3] 팽창지수가 6 : 3 : 1 일 때

[예시 4] 팽창지수가 60 : 30 : 10 일 때

이처럼 팽창지수는 자식 요소의 grow값 / 자식 요소들의 grow값의 총합 의 비율로 빈 공간을 가져갑니다.

[예시 3]의 box1은 box1의 팽창지수 6 / 팽창지수의 총합 6 + 3 + 1 = 3/5 만큼의 공간을 가져가고,

[예시 4]의 box1은 box1의 팽창지수 60 / 팽창지수의 총합 60 + 30 + 10 = 3/5 만큼의 공간을 가져갑니다.

팽창지수의 절대적 크기가 아닌 **총합에서의 비율**로 빈 공간을 차지하게 되는 것이죠.

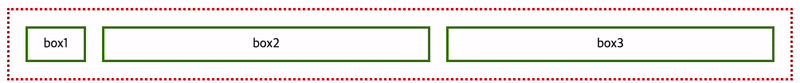
### 2. shrink : 자식 박스는 얼마나 줄어들 수 있을까요?

shrink(수축 지수)는 grow와 반대로, 설정한 비율만큼 박스 크기가 작아집니다. 비율이 클수록 더 많이 줄어드는 것이죠. 그러나 flex-grow 속성과 flex-shrink 속성을 함께 사용하는 일은 추천하지 않습니다. 비율로 레이아웃을 지정할 경우 flex-grow 속성 또는 flex: <grow> 1 auto와 같이 grow 속성에 변화를 주는 방식을 권장합니다. flex-shrink 속성은 width 나 이후 설명할 flex-basis 속성에 따른 비율이므로 실제 크기를 예측하기가 어렵기 때문입니다. **flex-grow 속성으로 비율을 변경하는 경우, flex-shrink 속성은 기본값인 1로 두어도 무방합니다.**

### 3. basis : 이 박스의 기본 크기는 얼마일까요?

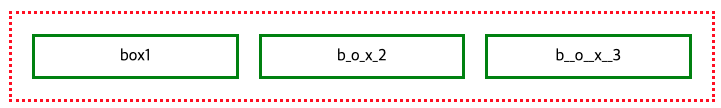
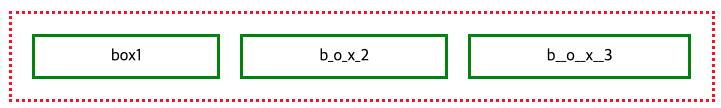
basis(기본 크기) 는 자식 박스가 flex-grow 나 flex-shrink 에 의해 늘어나거나 줄어들기 전에 가지는 기본 크기입니다. flex-grow 가 0일 때, basis 크기를 지정하면 그 크기는 유지됩니다.

grow 는 0 : 1 : 1 로 설정하고, box1에 flex-basis : 50px 로 설정해보겠습니다.



box1의 크기가 늘어나거나 줄어들지 않고 50픽셀을 유지하는 것을 확인할 수 있습니다.

**질문** : 세 개의 박스 모두 flex-grow : 1 일 때, 위 예시는 flex-basis : auto, 아래 예시는 flex-basis : 0 으로 설정해주었습니다. 두 예시 중 실제로 1 : 1 : 1 의 너비를 갖는 것은 어느 예시인가요? 왜 이런 차이가 발생할까요? 이번에는 직접 flex-basis 를 키워드로 구글링하여 답을 찾아보세요!



기억하세요. flex-grow 속성의 값이 0인 경우에만 flex-basis 속성의 값이 유지됩니다. diplay 속성에 flex 가 적용된 컨테이너 내부에 존재하는 자식 박스는 경우에 따라, **basis 로 설정된 크기가 항상 유지되는 것은 아닙니다.** flex-grow 속성의 값이 양수일 경우, 늘어나면서 flex-basis 속성에 적용한 값보다 커질 수도 있습니다. 실제 레이아웃을 구현하면서 막히는 경우에는, 다음의 원리를 참고할 수 있습니다.

**참고**

* width와 flex-basis를 동시에 적용하는 경우, flex-basis가 우선됩니다.
* 콘텐츠가 많아 자식 박스가 넘치는 경우, width가 정확한 크기를 보장하지 않습니다.
* (flex-basis를 사용하지 않는다면) 콘텐츠가 많아 자식 박스가 넘치는 경우를 대비해, width 대신 max-width를 쓸 수 있습니다.

ㅇ CLI

: GUI가 없는 형태로, 예를 들어 수만개의 서버 컴퓨터마다 GUI적용을 하려고 I/O장치를 부착할 수없으니 CLI를 이용해 서버컴퓨터를 사용하는 것이다.

ㅇ 햇갈리는 명령어

- \ : 백슬래쉬는 이어지는 특수문자를 출력하는 용도로, 실제 터미널에 나타나지 않음

- cat : 이어지는 파일을 순서대로 읽고 터미널에 출력해준다.

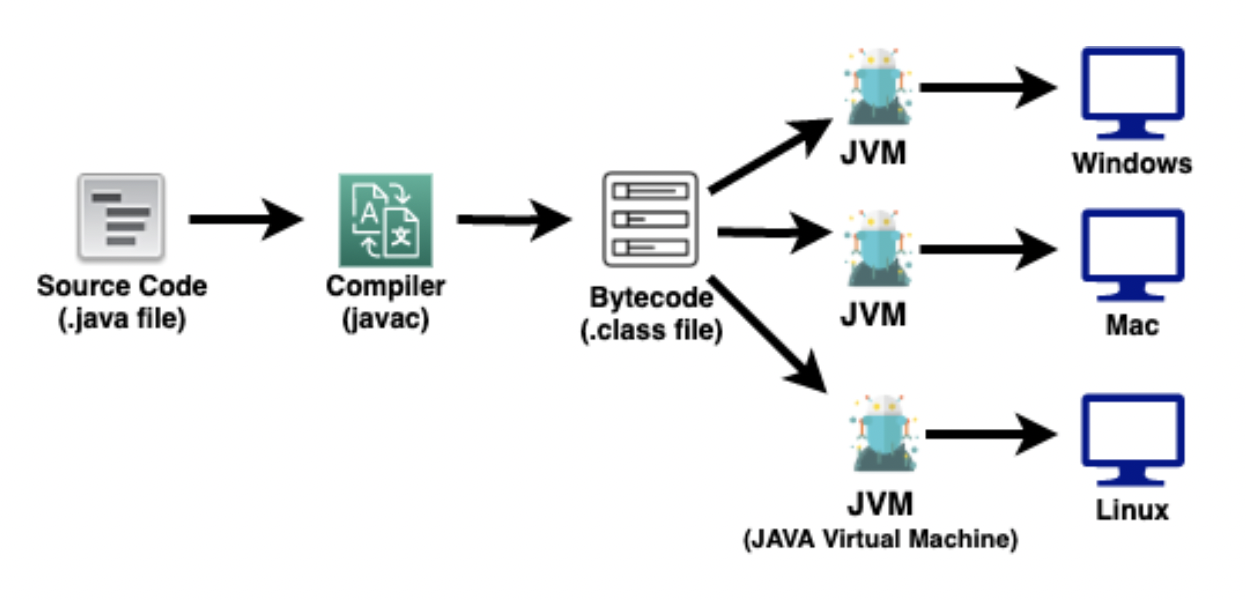
## **JVM(Java Virtual Machine)**

자바는 컴파일러(Compiler)를 통해 기계어([Machine Language](https://en.wikipedia.org/wiki/Machine_code))로 변환되는 언어입니다.

컴파일이란 특정 프로그래밍 언어를 기계가 이해할 수 있는 언어로 옮기는 번역 과정으로, 여기에서는 자바 언어를 JVM(Java Virtual Machine)이 이해할 수 있는 코드(바이트코드, Bytecode)로 변환하는 것을 의미합니다. JVM은 바이트코드를 운영체제에 맞는 기계어로 변환해줍니다.

JVM은 자바 프로그램을 실행시키는 도구입니다.

자바가 운영체제에 독립적인 것은 JVM이 있기 때문에 가능합니다. JVM은 자바 코드로 작성한 프로그램을 해석해 실행하는 별도의 프로그램입니다. 즉, 프로그램을 실행하는 프로그램입니다.



## **JDK(Java Development Kit)**

자바 설치와 관련하여, JDK와 JRE의 차이에 대해 먼저 알 필요가 있습니다. JDK와 JRE는 다음의 내용을 포함합니다.

* JRE(Java Runtime Environment) : JVM + 표준 클래스 라이브러리
* JDK(Java Development Kit) : JRE + 개발에 필요한 도구

만약 자바 프로그램을 실행만 할 것이라면 JRE만 설치해도 상관없습니다. 그러나 우리는 자바 프로그램을 개발할 것이기 때문에 JDK를 설치해야 합니다. JDK는 OracleJDK와 OpenJDK가 있습니다. 이 둘은 여러 차이가 있지만 지금은 OracleJDK는 오라클이라는 회사에서 관리하는 버전이고 OpenJDK는 오픈소스라는 것만 알아두어도 코드를 작성하는 데에는 문제가 없습니다.

# 자료구조

ㅇ 학습목표

- 알고리즘 문제에서 스택,큐 자료구조를 배열로 대체하여 흉내낼 수 있음

- 자료 구조의 개념과 구조를 파악하고 목적을 이해

- 알고리즘 문제의 각 상황에 맞는 자료구조를 떠올림

- 이진 탐색 트리를 이해

- BFS, DFS 개념을 이해하고 알고리즘 문제에 사용

- 자료구조를 활용하여 알고리즘 문제에 접근

1. 스택

- LIFO 구조 ( Latest In First Out )

- 데이터를 하나씩 넣고 뺄 수 있으며 하나의 입출력 방향을 가짐

ex.

1) 브라우저의 뒤로가기, 앞으로 가기 기능 구현

- 새로운 페이지 접속 시, 현재 페이지를 Prev Stack 에 보관

- 뒤로 가기 버튼 눌러 이전 페이지로 돌아갈 때,

현재 페이지를 Next Stack 에 보관하고

가장 나중에 보관된 페이지를 Prev Stack 의 현재 페이지로 가져옴

- 앞으로 가기 버튼 눌러 방문한 페이지로 이동 시 Next Stack 의 가장 마지막 보관된 페이지

가져옴

- 마지막으로 현재 페이지를 Prev Stack 에 보관

2. 큐의 구조

- FIFO 구조 ( First In First Out )

- 데이터를 하나씩 넣고 뺄 수 있으며, 두개의 입출력 방향이 있음

ex.

1) 문서를 작성하고, 출력 버튼을 누르면 해당 문서는 인쇄 작업 큐에 들어감

2) 프린터는 인쇄 작업 큐에 들어온 문서를 순서대로 인쇄

( 컴퓨터 출력버튼 - 기억 장치의 큐에 하나씩 들어옴 - 큐에 들어온 문서 순서대로 인쇄 )

\* 버퍼

: 각 장치간 속도의 차이/ 시간의 차이를 극복하기 위해 임시 기억장치의 자료구조로 큐를

사용

ㅇ 순회

1) 전위 (PreOrder) : 중간 - 왼 - 오

2) 중위 (inOrder) : 왼 - 중간 -오

3) 후위 (PostOrder) : 왼 - 오 - 중간

ㅇ 탐색

1) BFS : 가장 가까운 정점부터 탐색

( 너비 우선 탐색 )

ex.

한국에서 미국으로 가는 비행기를 예약할 때, 직항과 경유로 등 다양한 여정 중에서 최단 경로를 알아내야 할때 사용

2) DFS : 하나의 경로를 끝까지 탐색한 후, 미국 도착이 아니라면 다음 경로로 넘어가 탐색

( 깊이 우선 탐색 )

ex.

한국에서 출발하는 항공기의 모든 경로 중 미국에 도착하는 여정을 알아내고 싶을 때