《 _____ 영어 진로 세특 보고서 》

하이에듀

주제	관심 분야 영어로 소개하기 (코딩의 정의와 코딩 시 사용되는 언어)
가이드	**1. 서론** 학생의 희망 진로인 '컴퓨터 공학'에 맞춰 관심 분야를 '코딩'으로 설정하여 코딩이 무엇인지 영어로 소개하는 활동이다. **우리가 인간의 언어 중 하나인 영어를 통해 전 세계 사람들과 소통하기 위해 영어를 배우고 있듯이, 컴퓨터와 소통하기 위해서는 프로그래밍 언어를 배워야 하며 이 언어를 이용해 프로그래밍하는 과정을 코딩이라고 한다.** 이 정의를 더 자세히 알아보자. **2. 본론** (발표 내용) 컴퓨터에게 일을 시키기 위해서는 그 일의 순서를 미리 하나도 빠짐없이 컴퓨터에 가르쳐야 한다. 그 순서를 컴퓨터가 이해할 수 있는 언어로 기술한 것을 프로그램이라고 한다. 숫자나 기호 혹은 문자를 컴퓨터에서 처리할 수 있도록 사전에 정해놓은 일정한 규칙에 따라 프로그램을 작성하는 작업을 바로 '코딩'이라고 한다. 코딩은 논리적인 제어구조에 기반하여 수행해야 하므로, 코딩 과정을 통해 사고력이나 문제 해결 능력을 키울 수 있다. 이러한 측면에서 코딩은 필수 교육 중 하나로 자리잡기도 했다. 프로그래밍 언어는 매우 다양한데, 공통적으로 영어를 기반으로 하고 있다는 점이 아주 중요한 특징이다. (컴퓨터는 오로지 0과 1이라는 이진수만을 이해하는 기계이므로 중간에 어떠한 처리 과정이 존재해야 하지만) 0과 1만으로 이루어진 컴퓨터의 언어를 사람이 이해하기 어렵기 때문에 탄생한 것이 바로 어셈블리어이다. **3. 결론** 코딩을 소개하기 위해 발표 준비를 하는 과정에서 영어와 프로그래밍 언어의 유사점을 확인할 수 있었다. (자세한 내용은 자료3에 첨부합니다.) 우리가 전 세계 사람들과 소통하기 위해 영어를 공부하듯이, 컴퓨터와 잘 소통하기 위해 프로그래밍 언어를 열심히 공부해야겠다는 다짐을 얻을 수 있는 활동이었다.

자료1. 코딩이란?

코딩이란 요약하여 말하자면 '숫자나 기호 혹은 문자를 컴퓨터에서 처리할 수 있도록 사전에 정해놓은 일정한 규칙에 따라 컴퓨터 프로그램을 작성하는 작업'이다.

코드(Code)는 문자나, 숫자 등을 다른 방법으로 표현하기 위해 사전에 정의해 놓은 규칙으로, 코딩이란 이러한 코드를 일정한 규칙에 따라 번호 혹은 또 다른 문자 등의 값을 부여하는 과정을 의미한다. 소프트웨어 공학에서는 보통 컴퓨터에서의 자료 처리를 위한 컴퓨터 프로그램을 작성하는 작업을 의미한다.

컴퓨터 프로그램이 정상적으로 실행되기 위해서는 코딩된 프로그램에 대한 컴파일 등의 과정이 추가로 필요하나, 여기에서 코딩은 단순하게 컴퓨터 프로그램을 작성하거나 테스트하는 과정 자체를 의미한다. 또한, 컴퓨터 프로그래밍과 그 의미를 혼용하기도 하나, 컴퓨터 프로그래밍은 프로그램의 설계에서부터 코딩, 테스트 및 디버그에 이르기까지의 프로그램의 실행에 대한 전반적인 생애주기를 기술하는 과정인 데 반해, 코딩은 컴퓨터 프로그래밍의 한 부분이다.

(+ 프로그램이란?: 컴퓨터에 일을 시키기 위해서는 그 일의 순서를 미리 하나도 빠짐없이 컴퓨터에 가르쳐야 한다. 그 순서를 컴퓨터가 이해할 수 있는 언어로 기술한 것을 프로그램이라고 한다. 컴퓨터는 프로그램이 주어지면 프로그램에 기술된 명령대로 일을 하므로 만약 명령이 잘못되어 있으면 잘못된 대로 처리할 수 밖에 없다. 따라서 명령문인 프로그램은 모든 경우를 고려한 완전한 것이어야 한다.)

효율적인 코딩을 위해서는 프로그램의 흐름을 제어하기 위한 제어구조에 대해 이해하는 것이 중요하다. 제어구조란 프로그램의 실행 순서나 처리 흐름을 논리적으로 제어하기 위한 문장의 구조로써, 순차적 형태의 순차 구조, 주어진 조건에서 선택하는 선택 구조, 특정 구문을 반복해서 수행하는 반복 구조 등이 있다. 대표적인 프로그래밍 언어 중 하나인 C나 JAVA와 같은 프로그래밍 언어를 사용해서 개발한다고 가정했을 때, 프로그래밍 언어별로 문법은 서로 다르더라도 공통으로 제어구조에 기반을 두어 코딩하게 된다. 최근에는 직접 코딩하지 않고도 시각적인 블록 형태의 도구를 끌어놓기를 하는 것만으로 코딩이 가능하

도록 설계된 엔트리와 같은 개발도구도 활용되고 있다.

이렇듯 코딩은 논리적인 제어구조에 기반하여 수행해야 하므로, 코딩 과정을 통해 사고력이나 문제해결 능력을 키울 수 있다. 이러한 측면에서 코딩은 초등학교 필수 교육으로 자리 잡게 되었다. 또한 소프트웨어를 개발하는 기업에서는 지원 인력에 대한 능력을 검증하기 위해 채용 과정에 코딩 테스트를 도입해서 운영하기도 한다.

출처
https://terms.naver.com/entry.naver?docId=1159443&cid=40942&categoryId=32837

자료2. 코딩 시 사용되는 언어

<파이썬·C언어 다 영어, 프로그래밍 언어는 만국 공통어>
https://www.joongang.co.kr/article/25062051

코딩이 대세인 요즘, 취업을 준비하는 취준생이거나 중고등학생이라면 파이썬(Python)을 배우라는 말을 한 번쯤 들어 봤을 것이다. 파이썬은 현재 세계에서 가장 주목받는 프로그래밍 언어다. 우리나라 중고등학교에서도 채택하여 코딩 교육을 하고 있다. 문법이 간결하고 표현 방법이 인간의 사고체계와 많이 닮아서 학생들이 이해하기가 쉽다. 따라서 코딩을 쉽게 배울 수 있다는 장점이 있다. 코딩을 처음 접하는 인문 학도들도 많이 선택한다. 그렇다면 파이썬이 가장 쉬운 언어일까?

파이썬 개발자는 네덜란드인

쉽다, 누구나 할 수 있다는 파이썬의 특징이 간혹 코딩에 대한 근본적인 개념을 오해하게 하기도 하는 것 같다. 우리가 파이썬을 써서 코딩을 손쉽게 한다고 해도 컴퓨터가 이해하는 언어는 모두 '0'과 '1'이다. 파이썬은 인간의 언어와 비슷하지만, 컴퓨터가 이해하는 '0'과 '1'로 수렴하기 위해 여러 단계의 절차를 거치게 된다. 그 절차는 무엇일까?

파이썬처럼 컴퓨터가 이해할 수 있는 언어를 '프로그래밍 언어'라고 부른다. 이에 반해 사람이 사용하는 언어는 저절로 생겨났기 때문에 '자연어'라고 부른다. 저절로 생겨났다는 말

은 어느 한 사람이 만든 것이 아니라 많은 사람이 사용하다 보니 자연스레 언어가 된 것이고, 지금도 사용하는 사람들에 의해 조금씩 사용법이나 의미가 변하게 된다. 자연어는 문법도 복잡하고, 예외도 많고, 화자의 의도에 따라 다양하게 쓰이기도 한다. 이에 반해 프로그래밍 언어란 누군가가 의도를 갖고 인위적으로 만들어 낸 '인공언어'라고 할 수 있다.

이렇게 만들어진 프로그래밍 언어는 만든 사람 혹은 조직에 의해 적극적으로 홍보된다. 많은 사람이 사용하고 많이 활용될수록 좋은 언어가 되기 때문이다. 전 세계적으로 사용되는 프로그래밍 언어는 대부분 '영어'를 기반으로 제작되어 있다. 영어권 개발자들이 대부분인데다가, 비영어권 사람들조차 영어로 만들었다. 세계적으로 널리 쓰이길 원했기 때문일 것이다. 앞서 말한 파이썬 역시 네덜란드 사람에 의해 발명되었지만, 네덜란드어를 쓰지 않고 영어로 만들어졌다.

프로그래밍 언어가 다양하지만, 영어를 기반으로 하고 있다는 점은 아주 중요한 특징이다. 단언컨대 인류 역사상 전 세계가 이토록 하나의 언어로 통합된 적은 없었다. 그렇다. 이것은 모든 인류의 지식이 프로그래밍 언어를 중심으로 한 곳으로 모이고 있음을 의미한다. 최근 기술 발전 속도가 그 어느 시대보다 빠른 이유도 이 때문이라고 생각한다. 4차 산업혁명을 맞이하며 특정 기술을 구현하기 위해서 한국, 미국, 중국, 일본, 인도 등 다양한 국가의 개발자들이 한 가지 언어로 소통하고 있다. 바로 만국 공통어인 프로그래밍 언어를 통해서 말이다.

프로그래밍 언어는 영어로 되어 있지만, 문제는 컴퓨터가 영어를 이해하지는 못한다는 사실이다. 컴퓨터는 오로지 0과 1이라는 이진수만을 이해하는 기계니까. 따라서 중간에 어떤 처리 과정이 일어나야 검퓨터 이해를 하는데, 그것은 언어의 치환, 번역, 해석과 같은 일들이다.

지금은 믿기 어렵겠지만, 과거의 프로그래머들은 0과 1만으로 코딩을 했다. 초창기에는 키보드나 모니터 같은 입출력 장치도 없었다. 대신 '천공카드(punched card)'가 있었다. 이것에 구멍을 뚫어 컴퓨터에 집어넣는 것이 코딩이었다. 구멍이 뚫린 것은 1이고 구멍이 뚫리지 않은 것은 0을 의미했다.

중략

2세대 언어

문제는 0과 1만으로 표현된 기계어 코드를 사람이 쉽게 이해하기가 어렵다는 점이다. 이런 기계어에 질린 사람들이 0과 1보다 좀 더 인간의 언어에 가까운 방법이 필요하다는 것을 실감했다. 그래서 기계어를 대신해 탄생한 것이 바로 '어셈블리어(assembly language)'이다. 예를 들어, '더하기'를 의미하는 '111111'을 이제부터 'ADD'나 'PLUS'라고 부르기로 약속하는 것이다. 그러면 프로그래머는 '0001 ADD 0010'과 같이 코딩할 수 있다. '더하기'가 기계어로 무엇인지 외우고 다닐 필요가 없어지는 셈이다. 'ADD'라고 치면 '111111'로 자연스럽게 치환되니까 말이다. 이 어셈블리어는 1세대 언어인 기계어 다음에 탄생했기 때

문에 '2세대 언어'라고 불린다. 그런데 이 어셈블리어에도 기계어와 마찬가지로 치명적인 약점이 있었다. 만약 컴퓨터(CPU)가 바뀌거나 업그레이드가 된다면 코딩을 새롭게 해줘야 했다. 예를 들어, 인텔 CPU에서는 '111111'이 '더하기'를 의미하지만 삼성 CPU에서는 '빼기'를 의미할 수도 있기 때문이었다.

3세대 언어

C언어, 유닉스 코딩하다 탄생

벨 연구소의 연구원이던 켄 톰슨은 PDP-7이라는 컴퓨터에서 돌아갈 운영체제인 유닉스(UNIX)를 어셈블리어로 코딩하고 있었다. 그런데 한창 코딩을 하던 도중에 컴퓨터가 PDP-11이라는 최신 기종으로 업그레이드되는 불상사가 벌어졌다. PDP-7을 대상으로 작성하던 어셈블리어 코드를 PDP-11을 대상으로 다시 작성해야 하는 귀찮은 상황이 발생한 것이다. 결국 이 문제로 고심하던 중 동료였던 데니스 리치가 그 유명한 C언어를 개발하게 된다. C언어는 컴퓨터가 바뀌거나 업그레이드가 되어도 새로 코딩할 필요가 없는 언어였다. 그 이유는 '번역기'라는 것이 중간에 존재했고, 그 번역기가 C언어 코드를 기계어 코드로 번역하도록 개발되었기 때문이다. 그리고 리치는 톰슨과 함께 자신이 개발한 C언어를 사용해서 유닉스를 개발하게 된다. 우리는 이러한 C언어를 '3세대 언어'라고 부른다.

사실 C언어는 유닉스를 만들기 위해 탄생한 언어였다. C언어와 유닉스는 이후 컴퓨터 역사에 지대한 공헌을 하게 된다. C언어로부터 C++, C#, JAVA, JavaScript, PHP와 같은 수많은 파생 언어가 탄생했고, 유닉스로부터 리눅스, 안드로이드, macOS, iOS 같은 수많은 파생 운영체제가 만들어질 수 있었다. 결국 안드로이드폰과 아이폰에서 사용되는 모든 운영체제(OS)의 원형이 다 이때 만들어진 셈이다. 앞서 언급했던 파이썬도 마찬가지로 C언어와 같은 3세대 언어라고 할 수 있다.

그렇다면 3세대 언어보다 더 높은 수준의 언어도 있을까? 다시 말해서 파이썬보다 더 쉽게 코딩을 할 수 있는 언어도 있을까? 지금도 그런 언어를 개발하기 위해 누군가는 부단히 노력하고 있으리라 믿는다. 미래의 어느 순간에는 인간이 대충 말해도 번역기가 그 의도를 제대로 해석해서 컴퓨터에게 전달해 줄 수 있다면, 인간은 더는 프로그래밍 언어를 배우기 위해 고생할 필요가 없게 된다. 이렇게 생각해 보면 가장 이상적인 세상이란 컴퓨터와 인간이 자유롭게 소통하는 세상일지도 모르겠다.

자료3. (추가) 영어와 프로그래밍 언어의 유사성

1. 구문적 유사성

1) 문장 구조: 영어와 프로그래밍 언어 모두 문장을 구성하는 기본 단위들이 있습니다. 영어에서는 주어, 동사, 목적어 등으로 이루어진 구문을 사용하여 문장을 구성합니다. 마찬가지로 프로그래밍 언어에서도 키워드, 식별자, 연산자 등의 요소들로 프로그램 문장을 구성합니다. 예를 들어, 영어에서 "The cat eats the fish"라는 문장은 프로그래밍 언어에서 "cat.eat(fish)"와 같이 표현될 수 있습니다.

2) 조건문: 영어에서는 "if-else" 문으로 조건을 나타내며, 프로그래밍 언어에서도 "if-else" 구문을 사용하여 조건에 따른 분기를 제어합니다. 예를 들어, 영어로 "If it's raining, take an umbrella; otherwise, leave it at home"라는 문장은 프로그래밍 언어에서 "if (rain) { takeUmbrella(); } else { leaveUmbrellaAtHome(); }"과 같이 표현될 수 있습니다.

3) **반복문:** 영어에서는 "while"이나 "for"와 같은 구문을 사용하여 반복을 나타낼 수 있으며, 프로그래밍 언어에서도 "while" 루프나 "for" 루프와 같은 구문을 사용하여 반복 동작을 수행합니다. 예를 들어, 영어로 "Keep studying until you understand"라는 문장은 프로그래밍 언어에서 "while (!understand) { keepStudying(); }"와 같이 표현될 수 있습니다.

4) **함수 호출:** 영어에서는 동사와 목적어의 조합을 사용하여 특정 동작을 수행하는 문장을 만들 수 있습니다. 프로그래밍 언어에서도 함수 호출을 통해 특정 동작을 수행합니다. 예를 들어, 영어로 "Print the message"라는 문장은 프로그래밍 언어에서 "print(message)"와 같이 표현될 수 있습니다.

이러한 예시들은 영어와 프로그래밍 언어의 구문적 유사성을 보여줍니다. 그러나 유의해야 할 점은 프로그래밍 언어는 보다 엄격한 문법 규칙을 가지고 있고, 문맥에 따라 해석되는 방식이 영어와 다를 수 있다는 점입니다.

2. 표현방법의 유사성

1) **변수와 명사:** 영어에서 명사는 사물이나 개념을 표현하는 데 사용됩니다. 마찬가지로 프로그래밍 언어에서 변수는 값을 저장하고 나중에 참조하기 위해 사용됩니다. 예를 들어, 영어로 "The cat is black"라는 문장에서 "cat"은 명사로 사용되며, 프로그래밍 언어에서 "cat = 'black'"라는 코드에서 "cat"은 변수로 사용됩니다.

2) **함수와 동사:** 영어에서 동사는 행동이나 동작을 나타내는 데 사용됩니다. 프로그래밍 언어에서 함수는 특정 작업을 수행하는 코드의 블록입니다. 예를 들어, 영어로 "The cat eats the fish"라는 문장에서 "eats"는 동사로 사용되며, 프로그래밍 언어에서 "cat.eat(fish)"라는 코드에서 "eat"는 함수로 사용됩니다.

3) **조건문과 부사:** 영어에서 부사는 동사나 형용사의 동작이나 특성을 더 정확하게 설명하는 데 사용됩니다. 프로그래밍 언어에서 조건문은 특정 조건에 따라 코드의 실행을 제어합니다. 예를 들어, 영어로 "If it's raining, take an umbrella"라는 문장에서 "If"는 조건을 나타내는 부사로 사용되며, 프로그래밍 언어에서 "if (rain) { takeUmbrella(); }"라는 코드에서 "if"는 조건문으로 사용됩니다.

4) **반복문과 전치사:** 영어에서 전치사는 위치, 방향, 시간 등을 나타내는 데 사용됩니다. 프로그래밍 언어에서 반복문은 특정 동작을 반복해서 수행합니다. 예를 들어, 영어로 "Repeat the process for each item"라는 문장에서 "for"는 전치사로 사용되며, 프로그래밍 언어에서 "for (item in list) { process(item); }"라는 코드에서 "for"는 반복문으로 사용됩니다.

이러한 사례들은 영어와 프로그래밍 언어가 표현력을 활용하는 방식에서 유사함을 보여줍니다. 영어와 프로그래밍 언어 모두 다양한 언어 요소를 조합하여 의미를 전달하고, 이를 통해 복잡한 개념과 동작을 효과적으로 표현합니다.

3. 문맥의 유사성

영어와 프로그래밍 언어는 문맥을 이해하고 활용하는 측면에서도 몇 가지 유사점을 가지고 있습니다. 다음은 영어와 프로그래밍 언어가 문맥을 다루는 방식에서 유사한 사례 몇 가지를 설명합니다.

1) **변수와 대입문**

영어에서는 문맥에 따라 단어나 구문의 의미가 달라질 수 있습니다. 프로그래밍 언어에서도 변수와 대입문을 통해 값을 할당하고 이를 다른 문맥에서 참조할 수 있습니다. 예를 들어, 영어 문장 "He saw a cat. The cat was black."에서 "cat"은 첫 번째 문장의 명사로, 두 번째 문장에서는 변수와 비슷한 역할을 합니다. 프로그래밍 언어에서도 변수를 사용하여 값을 할당하고, 이를 다른 문맥에서 참조할 수 있습니다.

```
cat = "black" # 변수에 값 할당

def describe_cat():
print("The cat is", cat) # 변수 사용하여 값을 출력

describe_cat() # "The cat is black" 출력
```

2) 조건문과 논리 연산자

영어에서는 문맥에 따라 조건문과 논리 연산자를 사용하여 조건을 표현하고 판단합니다. 프로그래밍 언어에서도 조건문과 논리 연산자를 사용하여 문맥에 따라 다른 동작을 수행할 수 있습니다. 예를 들어, 영어 문장 "If it's raining, take an umbrella"에서 "if"와 "raining"은 조건을 표현하는 데 사용됩니다. 프로그래밍 언어에서도 조건문과 논리 연산자를 사용하여 문맥에 따라 다른 동작을 수행할 수 있습니다.

```
is_raining = True

if is_raining:
print("Take an umbrella")
else:
print("No need for an umbrella")

# "Take an umbrella" 출력
```

이러한 사례들을 통해 영어와 프로그래밍 언어가 문맥을 이해하고 활용하여 의미를 전달하며, 변수와 대입문, 함수와 매개변수, 조건문과 논리 연산자를 사용하여 문맥에 따라 다른 동작을 수행한다는 것을 알 수 있습니다.

참고

https://blog.naver.com/miraeinjae1297/223146715513