**II**

**1. 인공지능 개발 환경**

1. **개발 환경 소개**
2. Google Colab

Google에서 무료 서비스로 제공하는 Colaboratory(Colab)는 Google 사의 하드웨어에 액세스할 수 있는 주피터 노트북(Jupyter Notebook) 인터페이스를 제공한다. 해당 노트북은 중앙 처리 장치(CPU)로 계산을 수행하거나 특수 그래픽 처리 장치(GPU) 및 텐서 처리 장치(TPU)를 통해 가속화할 수 있는 Google에서 제공 및 유지 관리하는 Linux 기반 가상 머신에서 실행된다.

1. Colab 사용하기

* 먼저, Colab을 사용하기 위해서 구글 계정이 있어야 한다. 이후 Colab 사이트(<http://colab.research.google.com/>)로 접속하여 로그인을 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 로그인하면 지금까지 Colab을 사용해서 작업한 파이썬 파일들을 보여준다. Colab에 처음 접속한 사용자는 아래 그림과 같이 최근 사용 파일이 없을 것이다.

텍스트이(가) 표시된 사진

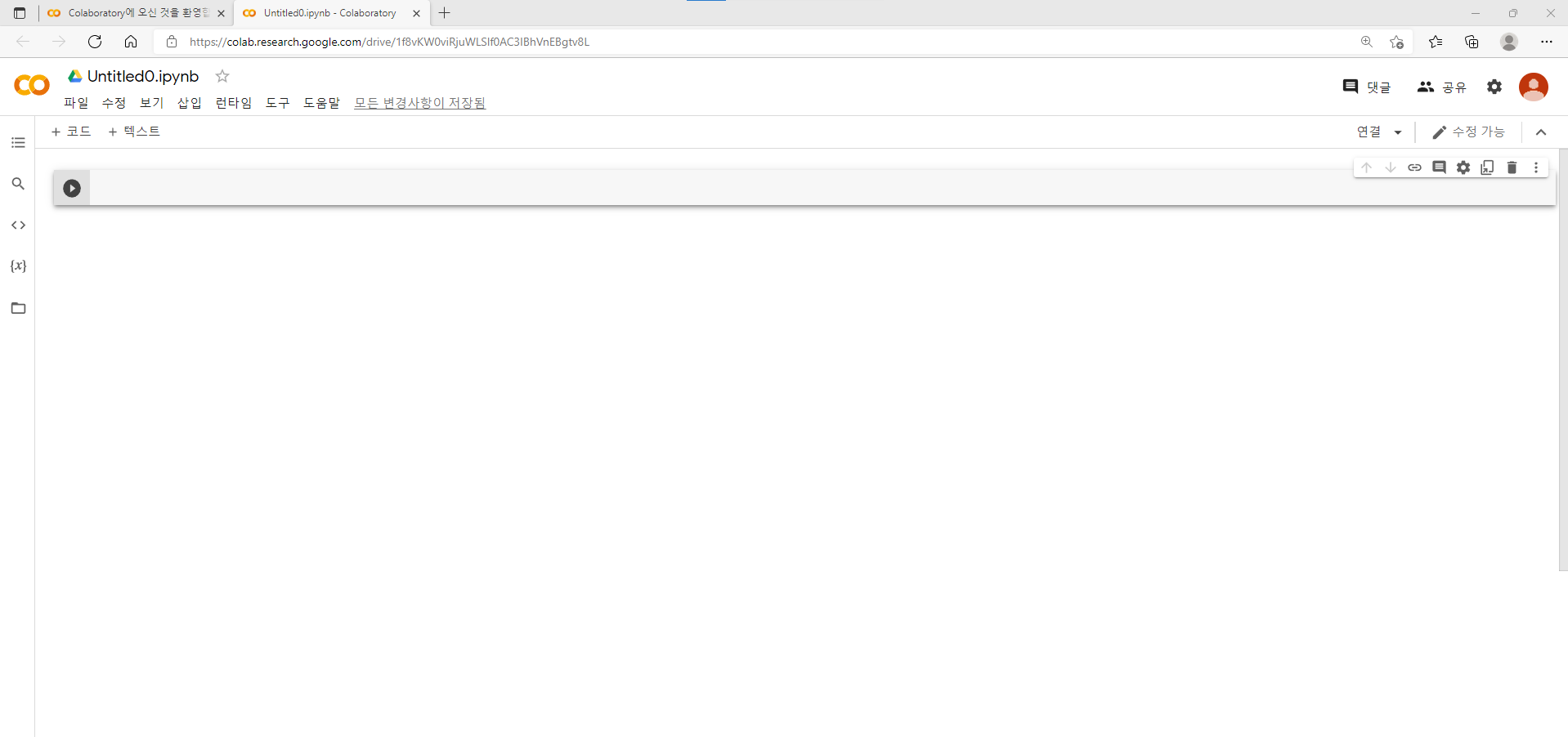
자동 생성된 설명

* 파이썬 코딩을 위한 새로운 노트북을 실행하려면 화면 왼쪽 상단에 있는 파일 메뉴를 클릭한다, 이후 하위 메뉴들 중 새노트 메뉴를 선택하면 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* 새노트를 실행하면 주피터 노트북과 유사하게 파이썬 코드를 입력하고 실행할 수 있는 노트북이 열리고 아래 그림과 같이 코드를 기입하고 실행할 수 있는 셀(cell)이 생성된다.



* Colab은 일반적인 개발 환경과 달리 로컬에서 파일을 불러올 수 없고 Google Drive를 이용해야 한다. Google Drive와 연동하기 위해서는 다음과 같은 두가지 중 하나의 방법을 따른다.

방법 1.

1. 왼쪽의 폴더 모양 – 드라이브 마운트 버튼을 눌러준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Google Drive 파일 액세스 허용

텍스트이(가) 표시된 사진

자동 생성된 설명

방법 2.

1. Google Drive를 마운트 하는 명령어(소스코드)를 입력한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 화면에 표시된 링크로 접속해 연결할 Google Drive 아이디로 로그인하고 허용을 누른다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Google Drive 마운트 허용 코드를 복사하여 Authorization Code 입력 부분에 붙여 넣은 후 연결한다.
2. **파이썬 기초**
3. 기초자료형(숫자, bool), 변수, 연산자

숫자형은 숫자 형태로 이루어진 자료형을 말한다. 우리가 흔히 사용하는 숫자형으로는 정수, 실수, 복소수가 있고 드물게 사용하는 8진수, 16진수가 있다. 불린 값은 True와 False 중 하나이고, 수학 계산에서는 각각 1과 0으로 평가를 한다.

|  |
| --- |
| >>> a = 267  >>> b = 45.12  >>> c = 0o172  >>> d = 0x8ff  >>> print (3 > 4)  False  >>> print(True or False)  True |

변수 a와 b는 각각 정수형과 실수형을 나타내며, 변수 c와 d는 8진수와 16진수를 나타낸다. 변수를 8진수(Octal)로 선언하기 위해서는 숫자가 0o(숫자 0과 알파벳 소문자 or 대문자 O)로 시작하면 된다. 16진수는 0x로 시작하면 된다.

숫자형은 다양한 연산자를 사용하여 계산을 할 수 있다.

|  |
| --- |
| >>> a = 7  >>> b = 2  >>> a + b  9  >>> a \* b  14  >>> a / b  3.5  >>> a \*\* b  49  >>> b % a  2  >>> a // b  3 |

일반적인 사칙연산(+, -, \*, /)은 한 번쯤은 접했을 연산자일 것이다. 이외에도 제곱을 나타내는 \*\* 연산자, 나눗셈의 나머지 값을 돌려주는 % 연산자, 나눗셈의 몫 값을 돌려주는 // 연산자가 있다.

변수는 값의 이름이다. 알파벳 대소문자 모두 사용 가능하며, 숫자도 변수명의 일부로 사용이 가능하지만 맨 처음에 숫자를 사용할 수는 없다.

|  |
| --- |
| Apple = 451 # 가능  Apple1 = 451 # 가능  2Apple = 451 # 불가 |



















1. 조건문(if), 반복문(while, for)

프로그램을 작성하면서 제어문을 사용하는 경우가 있다. 제어문은 크게 if, while, for으로 구성된다. 조건을 판단하여 해당 조건에 맞는 상황을 수행하는데 쓰는 것이 if 문이다.

|  |
| --- |
| >>> James = 77  >>> Jessica = 54  >>> if James > Jessica:  … print(‘James is heavier than Jessica’)  … elif James == Jessica:  … print(‘James is equal to Jessica’)  … elif James < Jessica:  … print(‘James is lighter than Jessica’)  …  James is heavier than Jessica |

if 문은 조건식이 True인 경우 명령을 실행하며, False일 경우 명령문은 건너뛰고 다음 코드를 실행한다. if 문은 단일, 2중, 다중 형식으로 구성할 수 있으며, elif 문을 사용하면 이전 조건이 거짓일 때 수행되고 조건을 여러 개 주는 것이 가능하다. 여기서 == 연산자는 일치하는가를 물어보는 연산자이다.

**Q. if문을 사용하여 3개의 숫자 (9, -4, 6) 를 오름차순으로 정렬하여라.**

|  |
| --- |
| number1, number2, number3 = 9, -4, 6  if number 1 > number2:  number1, number2 = number2, number1  if number2 > number3:  number2, number3 = number3, number2  if number1 > number2:  number1, number2 = number2, number1  print(“The sorted numbers are”, number1, number2, number3) |

인접한 숫자의 배열이 내림차순인 조건을 if 문으로 선별해 숫자의 배열을 바꾸어 저장해주는 방식으로 숫자의 배열을 오름차순으로 정렬하고 있다.

반복문 while 문은 조건문이 참이 될 때까지 반복해서 실행시킨다. 경험치가 10을 달성하면 레벨이 오르는 게임을 만든다면 다음과 같다.

|  |
| --- |
| >>> EXP = 0  >>> while EXP < 10:  … EXP = EXP +1  … print(“경험치가 %d입니다.” % EXP)  … if EXP == 10:  … print(“레벨업!”)  …  경험치가 1입니다.  경험치가 2입니다.  …  경험치가 8입니다.  경험치가 10입니다.  레벨업! |

여기서 while 문의 조건문은 EXP < 10이다. 10보다 작으면 계속 반복해서 수행하며, EXP = EXP +1은 EXP 값을 계속 1씩 증가시킨다. 그리고 나서 EXP < 10 조건문이 거짓이 되면 while 문을 빠져나가게 된다.

**Q. while 문과 덧셈기호를 이용해 곱셈기호 없이 두 수의 곱셈을 구현해보자.**

|  |
| --- |
| m = 3  n = 7  res = 0  while n>0:  res += m  n = n – 1  print(‘m \* n =’, res) |

n 과 m 의 곱셈을 구현하기 위해 n이 양수인 조건 하에서 while 문을 반복하며 res 값에 m씩 더하여 저장한다. while 문이 반복될 때마다 n의 값은 1씩 빼서 저장하면, 정확히 n번 반복되는 결과를 가져온다.

다른 반복문으로 for 문이 있다. while 문과 비슷한 반복문인 for 문은 유용하고 문장 구조 이해가 쉽다는 장점이 있다. 다음 for 문의 간단한 예시를 보며 이해를 돕고자 한다. 10명 학생이 운전면허 필기 시험을 보았다. 시험 점수가 70점이 넘으면 합격이고 그렇지 못하면 불합격이다.

|  |
| --- |
| exams = [66, 70, 76, 56, 80, 64, 74, 90, 50, 62]  number = 0  for result in exams:  number = number +1  if result >= 70:  print(“%d번 응시자는 합격입니다.” % number)  else:  print(“%d번 응시자는 불합격입니다.” % number) |

각각의 응시자에게 번호를 부여하기 위해 number 변수를 사용하였다. 리스트 exams에서 차례로 점수를 result 변수로 받아 for 문 안의 문장을 수행한다. 수행이 한 번씩 진행되면서 number는 1씩 증가한다.

**Q. for 문을 이용하여 양의 정수가 소수인지 아닌지 판별해보자.**

|  |
| --- |
| n = 9  for a in range (2,n):  if n % a ==0 :  print(‘not prime’)  break;  eles:  print(‘prime’) |

‘for a in range (2,n)’ 문 이하에서 a 자리에 2부터 n-1까지의 수를 넣어가며 한번씩 반복문을 수행한다. N을 2부터 n-1까지의 수로 나눠보았을 때, 하나의 수라도 n을 나누어 떨어지게 하는 수가 있으면 n은 소수가 아니다. 모두 n을 나누어 떨어지게 하지 못한다면 n은 소수이다.

1. 함수

프로그래밍을 진행하면 같은 내용을 반복해서 작성하는 경우가 발생한다. 이때 함수를 사용하면 편리하게 작성할 수 있다. 또한, 프로그램 흐름 파악 및 오류가 어디서 발생하는지도 바로 알아차릴 수 있다.

|  |
| --- |
| def multiply(a, b):  result = a \* b  return result |

위는 일반적인 함수의 예이다. multiply는 함수의 이름이며, a와 b는 함수에 사용되는 매개변수이다. 즉, multiply 함수는 2개의 입력 값을 받아 서로 곱한 값을 돌려준다.

다음은 여러 개의 입력 값을 받는 함수에 대해서 알아본다.

|  |
| --- |
| def multiply\_many(\*parameters):  result = 0  for i in parameters:  result = result + i  return result |

위에서의 multiply\_many 함수는 입력 값이 몇 개이든 상관이 없다. \*parameters처럼 매개변수이름 앞에 \*을 붙이면 입력 값을 전부 모아 튜플로 만들어주기 때문이다.

**Q. 두 수의 최대공약수를 구하는 함수를 만들어보자.**

|  |
| --- |
| def gcd(a,b) :  while b > 0 :  r = a % b  a = b  b = r  return a |

b가 양수일 조건 하에서, while 문이 반복되고 반복문을 거칠 때마다 a와 b를 b와 r (r은 a를 b로 나눈 나머지)로 업데이트 한다. a를 b 로 나눈 나머지가 0이 될 때, 반복문은 멈추며, 마지막으로 업데이트 된 a 값이 a 와 b의 최대 공약수이다.























































































1. 시퀀스 자료형(문자열, 리스트, 튜플)

시퀀스 자료형에는 문자열, 리스트, 튜플이 있다. 문자열은 문자, 단어 등으로 구성된 문자들의 집합이다.

|  |
| --- |
| >>> a = ‘home’  >>> a[0]  ‘h’  >>> a[1:3]  ‘om’  >>> a[:4]  ‘home’  >>> a[1:]  ‘ome’  >>> a = ‘c’ + a[1:]  >>> a  ‘come’ |

문자열에서는 한 글자마다 번호를 매기게 된다. 맨 앞에서부터 0부터 순차적으로 번호가 매겨진다. a[:4]는 a[4]는 포함하지 않고, a[1:]는 a[1]를 포함하여 출력한다. 시퀀스의 일부를 취하는 것을 슬라이싱이라고 한다.

리스트를 만들 때는 대괄호([])로 감싸주고 각 원소는 쉼표로 구분한다.

|  |
| --- |
| >>> a = []  >>> b = [1, 2, 3]  >>> c = [‘Process’, ‘Engineering’, ‘Systems’]  >>> d = [1, 2, ‘Chemical’, ‘Engineering’] |

리스트는 a처럼 아무것도 포함하지 않을 수 있고, b처럼 숫자를 원소로 가질 수 있다. 또한, c처럼 문자열로 구성될 수도 있으며, d처럼 숫자와 문자열을 함께 가질 수도 있다.

|  |
| --- |
| >>> example = [2, 6, 8]  >>> example.append(4)  >>> example  [2, 6, 8, 4]  >>> example.sort()  >>> example  [2, 4, 6, 8]  >>> example.insert(0, 0)  >>> example  [0, 2, 4, 6, 8]  >>> del example[3]  >>> example  [0, 2, 4, 8] |

위 예를 보면, append는 example에 원소 4를 추가하는 기능하고 sort 함수를 이용하면 example을 원소 크기 순으로 정렬할 수 있다. 여기에 원하는 순서에 원하는 원소를 넣고자 한다면, insert 기능을 사용할 수 있으며, 원소를 삭제하고자 한다면 del 함수를 사용하면 된다.



리스트는 아래처럼 중첩하여 또 다른 리스트를 포함할 수 있다. 이와 같은 형태의 리스트를 중첩 리스트(nested list)라고 부른다.

|  |
| --- |
| >>> my\_list = [“Artificial”, [3, 6, 9], [“Intelligence”, 4, 8]]  >>> print(my\_list[0])  Artificial  >>> print(my\_list[1])  [3, 6, 9]  >>> print(my\_list[1][2])  9  >>> print(my\_list[0][0])  A  >>> print(my\_list[0][2])  t  >>> print(my\_list[2])  [“Intelligence”, 4, 8]  >>> print(my\_list[2][0])  Intelligence |

**Q. 중첩 리스트의 모든 원소들의 합을 구하는 nested\_sum이라는 함수를 만들어보자.**

|  |
| --- |
| def nested\_sum(nestedList, newList = [0]):  def flatlist(nestedList):  for I in range(len(nestedList)):  if type(nestedList[i]) == int:  newList.append(nestedList[i])  else:  flatlist(nestedList[i])  return newList  flatlist(nestedList)  print(sum(newList)) |

먼저, 중첩 리스트에 들어있는 리스트 원소들의 포함 원소들을 모두 꺼내 하나의 리스트 안의 원소들로 만들어주는 ‘flatlist’ 함수를 정의하고, ‘flatlist’ 함수를 사용해 중첩 리스트를 하나의 리스트로 만들어 sum 함수를 통해 원소들의 합을 구한다.

마지막으로 튜플은 몇 가지 점을 제외하면 리스트와 비슷하다. 리스트는 []으로 둘러싸이지만 튜플은 ()로 둘러싸인다. 그리고 리스트는 값의 생성, 삭제, 수정이 가능하지만, 튜플은 값을 바꾸거나 삭제할 수 없다.

|  |
| --- |
| >>> s1 = (4, 7, ‘p’, ‘q’)  >>> del s1[0]  텍스트이(가) 표시된 사진  자동 생성된 설명 |

튜플의 원소를 리스트처럼 del을 이용해서 지우려고 하면 위와 같이 오류가 발생함을 알 수 있다. 또한, 튜플의 원소를 변경하려고 한다면 아래와 같은 오류가 발생한다.

|  |
| --- |
| >>> s1 = (4, 7, ‘p’, ‘q’)  >>> s1[0] = ‘a’  텍스트이(가) 표시된 사진  자동 생성된 설명 |