

임상의를 위한 AI 교육 – 기초과정 1주차

딥러닝을 위한 Python 기초

서울대학교병원 융합의학과 김영곤 교수



※ 본 수업자료는 “파이썬 딥러닝 머신러닝 입문” 학습서
기반으로 제작되었습니다.

Part 1. 개발환경 설정

1. 구글 코랩(Colab)이란?



- 파이썬 데이터 분석용 무료 코드 에디터
- 웹 브라우저에서 바로 실행 가능
- 주피터 노트북(Jupyter Notebook) 기반

장점	제약 사항
<ul style="list-style-type: none">•설치가 필요 없다.•인터넷 접속되는 컴퓨터만 있으면 사용 가능•머신러닝, 딥러닝에 필요한 GPU 환경 지원 (그래픽카드 없어도 가능)•무료 사용	<ul style="list-style-type: none">•무료 계정의 경우, 사용상 제약이 있음<ul style="list-style-type: none">- 12시간이 넘으면 세션이 종료 된다.- 메모리(RAM)와 저장 장치 용량에 제한• 제한 없이 사용하려면 유료 서비스 구독 필요

Part 1. 개발환경 설정

2. 코랩 시작하기

2-1

구글 계정 로그인

- 웹브라우저에 구글 주소(www.google.com)를 입력하고 [로그인] 버튼을 누른다. 구글 계정을 선택 한다. 계정이 없으면, 새로운 계정을 만든다.
- 계정 비밀번호를 입력한다. 로그인이 완료되면, 프로필 이미지가 표시된다.



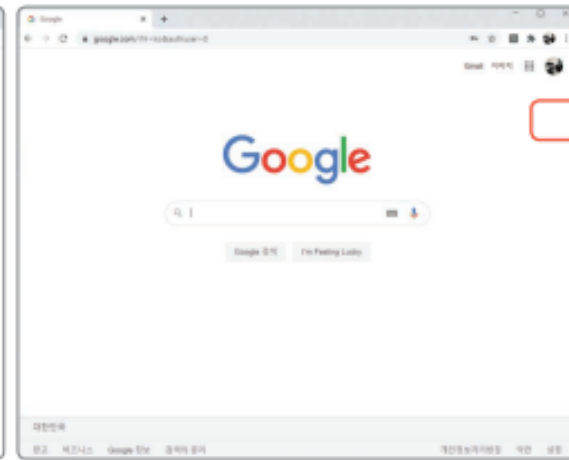
[그림 1-1] 구글 홈페이지 로그인



[그림 1-2] 구글 계정 선택



[그림 1-3] 비밀번호 입력



[그림 1-4] 로그인 완료

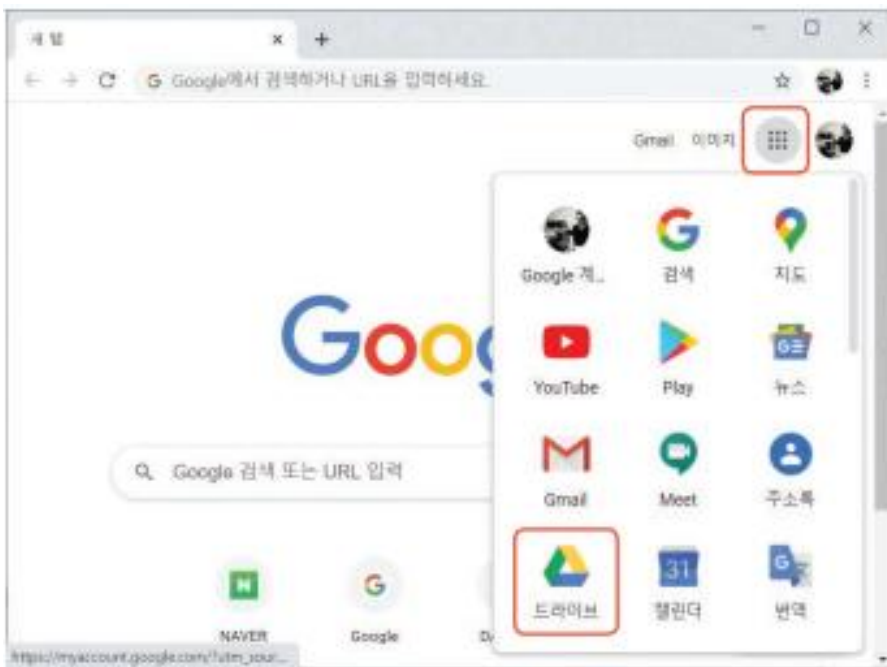
Part 1. 개발환경 설정

2. 코랩 시작하기

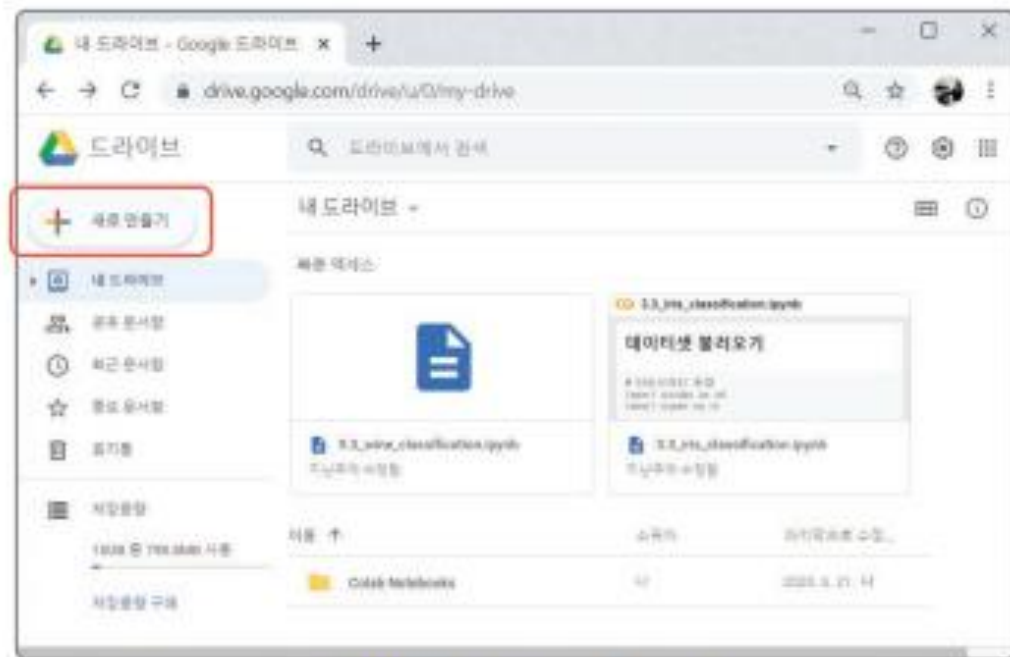
2-2

구글 코랩 실행하기

- 화면 오른쪽 위의 "Google 앱" 메뉴를 클릭하면 다음과 같이 실행 가능한 앱이 표시된다. 여기서 "드라이브" 앱을 찾아 선택한다.
- 다음과 같이 구글 드라이브가 실행되면 왼쪽 메뉴에서 [새로 만들기] 버튼을 누른다.



[그림 1-5] 구글 드라이브 실행



[그림 1-6] 구글 드라이브 메인 화면

Part 1. 개발환경 설정

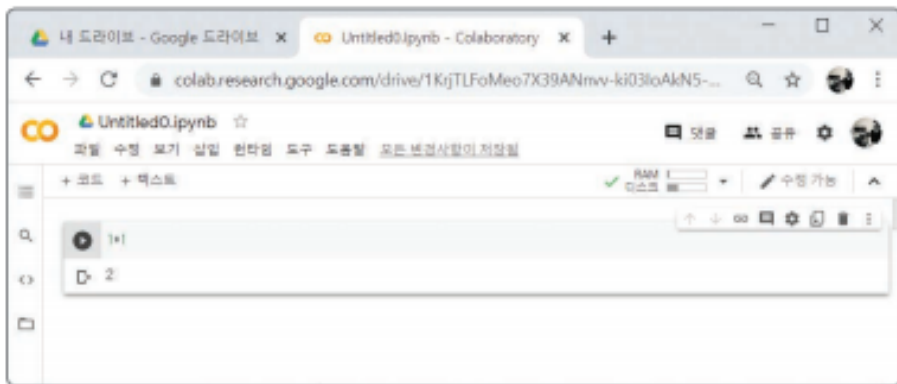
2. 코랩 시작하기

2-3

“Hello, Colab” 코딩하기

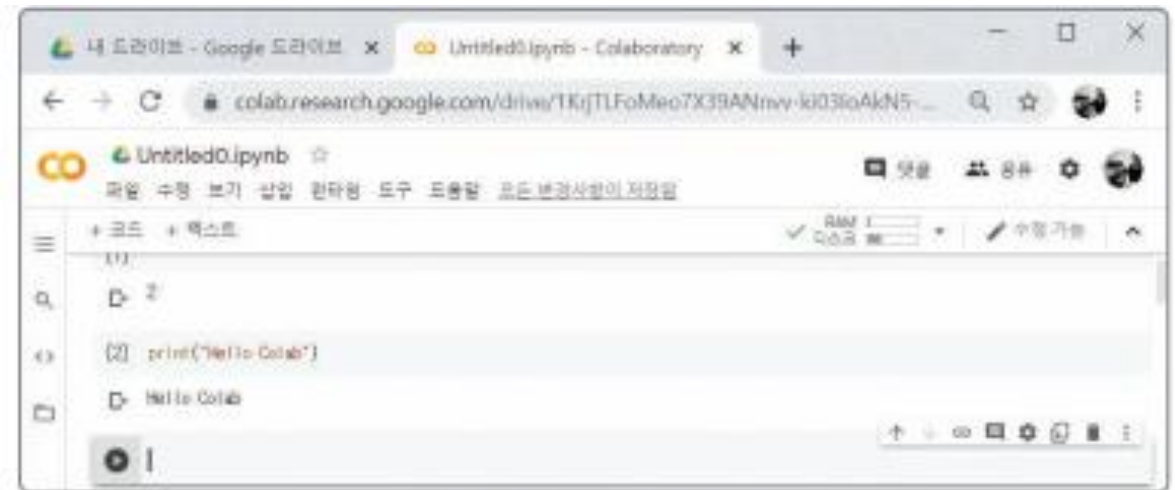
- 코드 셀에 다음과 같이 1+1을 입력하고 실행한다.

```
<스> 1.1_hello.ipynb  
[1] 1+1
```



[그림 1-9] 덧셈 연산 실행 결과

- "Hello Colab"을 화면에 출력해 본다. print 명령을 이용하면 괄호 안에 입력한 데이터를 화면에 출력할 수 있다.



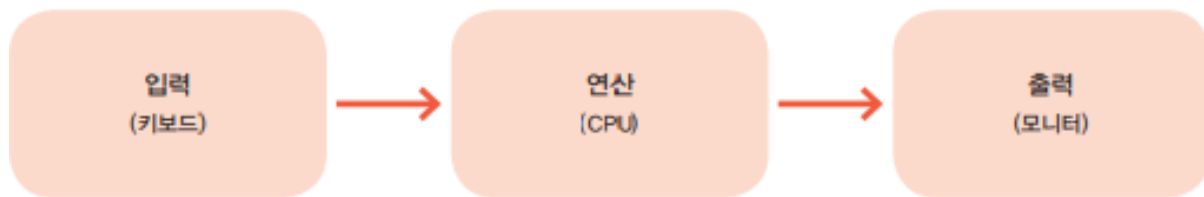
[그림 1-10] print 명령 실행 결과

1. 프로그래밍 기본 개념

1-1

데이터 입력과 출력

- 컴퓨터는 연산에 필요한 데이터를 키보드 등 입력 장치를 통해 전달받고, 연산 장치(CPU, GPU)에서 계산한 결과를 모니터, 프린터와 같은 출력 장치를 통해 보여준다.



[그림 2-1] 데이터의 입력, 연산, 출력 프로세스

[입력 단계] 키보드를 이용하여 1+2를 입력한다. 1과 2는 데이터(또는 자료)라고 부르고, 더하기 (+) 연산 기호를 연산자라고 말한다.

[연산 단계] 코드 셀 왼쪽의 실행() 버튼을 누르면, 컴퓨터는 입력값을 읽어 덧셈 연산을 처리한다.

[출력 단계] 코랩은 실행 결과를 코드 셀 아래 쪽에 표시한다. 1과 2를 더한 값인 3이 출력된다.

<소스> 2.1_input_processing_output.ipynb

[1] 1+2

3

Tip 다운로드 받은 예제 파일들은 가능하면 참고용으로 활용하고, 새 노트북에 직접 코드를 입력해 보는 것이 좋다.

1. 프로그래밍 기본 개념

1-2

변수에 저장

- 변수를 이용해 메모리 에 저장된 값을 불러올 수 있다. 이 과정을 '어떤 값을 변수에 할당'한다고 말한다.
- 숫자 5를 변수에 할당해 보자. 변수 이름은 자유롭게 정할 수 있는데, 다음의 예제에서는 var1이라는 변수 이름을 사용하고 있다. 여기서 등호(=)는 '좌변과 우변이 같다'는 뜻이 아니고, '우변에 있는 값(5)을 좌변의 변수(var1)에 할당한다'는 뜻이다.

```
[2] var1 = 5
```

- 변수 이름을 코드 셀에 입력하고 실행하면 변수에 할당되어 있는 값이 화면에 표시된다. 앞의 코드에서 var1 변수에 할당해 놓은 숫자 5가 화면에 출력된다.

```
[3] var1
```

```
5
```

- var1 변수에 다른 값을 할당하면 var1 변수의 값이 변경된다. 이처럼 저장하는 값이 변할 수 있다는 의미에서 변수(variable)라고 부른다.

```
[5] var2 = 20
```

```
var2
```

```
20
```


1. 프로그래밍 기본 개념

1-3

화면에 출력

- 파이썬 내장 함수인 print 명령을 사용하여 화면에 출력한다. print 함수의 괄호 안에 화면에 표시하려는 대상을 입력한다.
- var1 변수를 입력하면 변수가 저장하고 있는 값인 10을 출력한다.

```
[7] print(var1)
```

```
➡ 10
```

- 변수를 대신하여 연산식(var1 + var2)을 입력하는 것도 가능하다. 덧셈 결과인 30이 출력된다.

```
[8] print(var1 + var2)
```

```
➡ 30
```

- 쉼표(,)로 구분해서 여러 객체를 입력하면 서로 한 칸 간격을 띄우고 출력된다.
- 예제의 print (var1, var2) 코드는 var1 변수의 값과 var2 변수의 값을 한 칸 띄우고 출력하라는 뜻이다. 숫자 10과 20이 한 칸 띄우고 출력된다.

```
[9] print(var1, var2)
```

```
➡ 10 20
```

- 파이썬에서 따옴표 안에 입력하는 글자 또는 숫자를 문자열로 인식한다. 큰 따옴표 (" ")와 작은 따옴표(' ') 모두 사용할 수 있다.
- 문자열을 print 함수에 입력하면 따옴표를 제외하고 따옴표 안의 문자열 값을 화면에 출력한다.
- print("덧셈:", var1 + var2)라는 코드는, "덧셈:" 문자열과 연산식(var1 + var2)의 결과값인 30을 한 칸 띄우고 출력하라는 뜻이다.

```
[10] print("덧셈:", var1 + var2)
```

```
➡ 덧셈:30
```

Tip 따옴표 안의 문자열 내용은 그대로 출력된다.

2. 자료형

2-1

숫자형

숫자형은 숫자가 갖는 특성을 프로그래밍 언어로 구현한 데이터 유형을 말한다.

숫자형 데이터는 정수형(int)과 실수형(float)으로 구분된다.

자연수와 정수는 정수형으로 정의하고, 소수점이 있는 숫자는 실수형으로 정의한다.

int
(정수형)

-3, -2, -1, 0, 1, 2, 3

float
(실수형)

3.14, 0.1, 10.5, 1.23456, 1e-6

[그림 2-2] 숫자형 데이터의 종류

2. 자료형

2-2

문자열

- 문자열(string)은 따옴표 안에 입력한다.
- 큰 따옴표(" ") 또는 작은 따옴표(' ') 안에 알파벳, 한글, 숫자, 공백(띄어쓰기), 기호 등을 입력한다.

<소스> 2.3_data_types_2.ipynb

```
[1] str1 = "Easy"
    str1
```

→ 'Easy'

- type 함수로 문자열의 자료형을 확인해 보면, string을 나타내는 str로 표시된다.

```
[2] print(type(str1))
```

→ <class 'str'>

- 따옴표 안에 문자와 숫자를 함께 사용해도 문자열이다.

```
[4] str3 = 'ver1.0'
    print(type(str3))
```

→ <class 'str'>

- str1 변수가 저장하고 있는 'Easy' 문자열은 E, a, s, y라는 4개의 문자를 원소로 갖는다.

```
[5] len(str1)
```

→ 4

2. 자료형

2-2-1

문자열 인덱싱

- 인덱스(index)는 어떤 배열 안에 원소가 위치하고 있는 순서를 나타낸다.
- 첫 번째 원소의 위치를 인덱스 0으로 하고, 두 번째 인덱스는 1이 된다. 이처럼 숫자 1씩 증가시켜 다음 위치를 나타내는 인덱스 숫자를 표시한다.

문자열 (길이 4)	E	a	s	y
인덱스 (0~3)	0	1	2	3

[그림 2-3] 문자열 인덱싱(앞에서부터 순서를 셀 때)

- 원소를 선택할 때는 대괄호([]) 안에 위치 순서를 나타내는 인덱스 숫자를 입력한다.

```
[6] str1[0]
```

```
→ 'E'
```

2. 자료형

2-2-1

문자열 인덱싱

- 인덱스 배열의 순서를 뒤에서 앞으로 표현할 수 있다. 끝 위치를 나타내는 인덱스는 -1로 정의한다.

문자열 (길이 4)	E	a	s	y
인덱스 (-4~-1)	-4	-3	-2	-1

[그림 2-4] 문자열 인덱싱(뒤에서부터 순서를 셀 때)

- 'Easy' 문자열의 마지막 원소인 "y"를 선택하기 위해서는 [-1]로 인덱스를 표시한다.

```
[8] str1[-1]
```

```
→ 'y'
```

- 뒤에서 3번째 원소인 "a"를 선택하려면 인덱스 -3을 사용한다.

```
[9] str1[-3]
```

```
→ 'a'
```

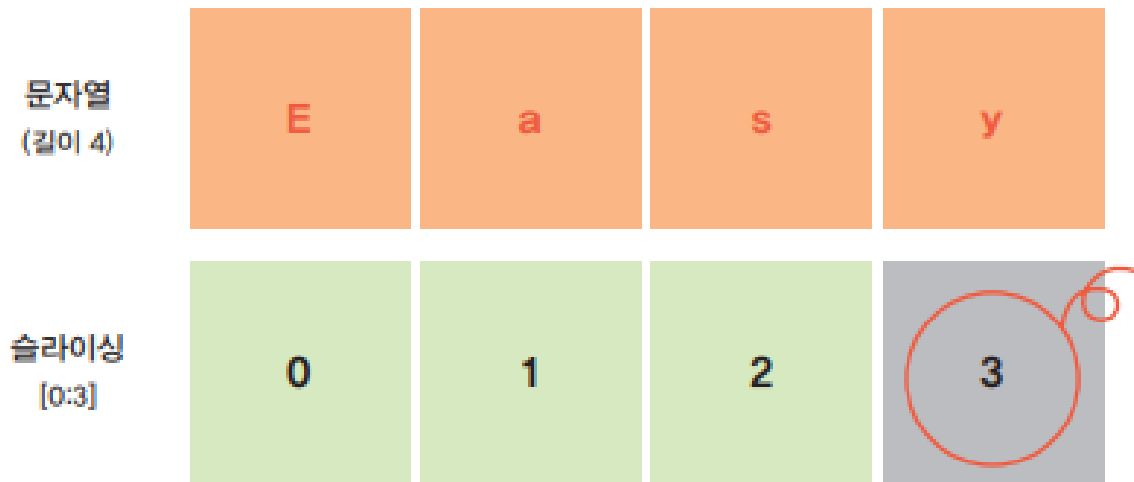
2. 자료형

2-2-2

문자열 슬라이싱

- 슬라이싱(slicing) 기법은 인덱스 범위를 지정하여 여러 개의 원소를 추출하는 방법이다.

[시작 인덱스:끝 인덱스] 형식으로 범위를 입력한다.



[그림 2-5] 문자열 슬라이싱

- 범위를 [0:3]으로 지정하면 3을 포함하지 않고 0, 1, 2 인덱스에 해당하는 "Eas"를 추출한다. 즉, 범위를 지정할 때 시작 인덱스는 포함하지만 끝 인덱스는 포함하지 않는다.

```
[10] str1[0:3]
```

```
→ 'Eas'
```

- 뒤에서부터 슬라이싱하는 경우에도 끝 인덱스 값은 포함하지 않는다.

```
[11] str1[-3:-1]
```

```
→ 'as'
```

2. 자료형

2-2-3

문자열 중간에 문자열 끼워 넣기

- %s는 문자열을 넣을 위치를 나타낸다.
- "%s 파이썬 딥러닝" 문자열에 str1 변수의 값을 %s 위치에 대신 끼워 넣는다.
- 즉, str1 변수의 값인 "Easy"가 %s 위치에 들어가 "Easy 파이썬 딥러닝"이라는 문자열을 만든다.



[그림 2-6] 문자열의 특정 위치에 끼워 넣기

2. 자료형

2-3

리스트

- 리스트(list)는 여러 개의 데이터를 하나의 모음(집합)으로 다루고 싶을 때 사용하는 대표적인 자료형이다.
- 리스트를 만들기 위해서는 대괄호([]) 안에 여러 개의 원소를 쉼표(,)로 구분하여 입력한다.

<소스> 2.4_data_types_3.ipynb

```
[1] list1 = [1, 2, 3, 4, 5]  
list1
```

```
[1, 2, 3, 4, 5]
```

- list() 함수에 배열을 입력하지 않으면 원소가 없는 빈 리스트를 생성한다.

```
[4] list3 = list()  
print(list3)
```

```
[ ]
```

Tip list 함수는 배열을 입력받아 리스트로 변환해 주는 명령이다.

- []와 같이 원소를 추가하지 않고 대괄호만 입력하는 방법으로 빈 리스트를 만들 수 있다

```
[5] list4 = []  
list4
```

```
[ ]
```


2. 자료형

2-3-1

리스트를 원소로 갖는 리스트

- 모든 종류의 파이썬 객체는 리스트의 원소가 될 수 있다.
- 따라서, 리스트도 다른 리스트의 원소가 될 수 있다.

```
[6] list_of_list = [list1, list2, list3]
```

```
list_of_list
```

```
→ [[1, 2, 3, 4, 5], [10], [ ]]
```

2-3-2

리스트 인덱싱

- 문자열의 인덱싱과 비슷하다. 인덱스(index)는 리스트 안에서 각 원소가 위치하고 있는 순서를 나타낸다.

[[1, 2, 3, 4, 5], [10], []]

인덱스 0

인덱스 1 인덱스 2

[그림 2-7] list of list 인덱싱

- 첫 번째 위치(인덱스 0)에 있는 원소를 선택한다. 원소 5개를 갖는 리스트(list1)이 출력된다.

```
[8] list_of_list[0]
```

```
→ [1, 2, 3, 4, 5]
```

2. 자료형

2-3-3

리스트 슬라이싱

- 리스트 원소의 위치(인덱스) 범위를 지정하여 여러 개의 원소를 추출한다. 문자열 슬라이싱 방법 과 비슷하다.
- 인덱스 범위를 [0:2]로 지정하면 0부터 1(범위의 끝인 2를 제외)까지 인덱스에 해당하는 원소가 선택된다.

```
[12] list_of_list[0:2]
```

```
➡ [[1, 2, 3, 4, 5], [10]]
```

2-3-4

리스트에 새로운 원소 추가

- 리스트의 맨 뒤에 새로운 원소를 추가할 수 있다.
- 리스트 객체에 도트(.) 연산자를 이용하여 append 메소드를 적용한다.

```
[13] list1.append(100)
```

```
list1
```

```
➡ [1, 2, 3, 4, 5, 100]
```

메소드(method)는 나중에 설명할 클래스 자료구조 내부에서 정의된 함수를 말한다. 클래스 자료에 어떤 기능을 처리할 때 사용하는 명령이다. 리스트 자료형은 클래스로 구현되어 있는데, append 메소드는 리스트에 새로운 원소를 추가하는 명령이다.

2. 자료형

2-4

튜플

- 튜플(tuple)은 리스트 자료구조와 형태가 비슷하다. 여러 원소를 순서대로 저장할 수 있고 각 원소를 쉼표(,)로 구분한다. 튜플은 () 안에 원소를 입력한다.

〈소스〉 2.5_data_types_4.ipynb

```
[1] tuple1 = (1, 2)
    tuple1
```

➡ (1, 2)

- 단, 리스트와 다르게 원소를 추가하거나 삭제할 수 없다는 점에서 다르다.

- 한편, 리스트와 마찬가지로 인덱싱이 가능하다.

```
[5] tuple2[0]
```

➡ 'a'

- 인덱스의 범위를 지정하여 선택하는 슬라이싱 방법도 적용 가능하다.

```
[6] tuple2[1:]
```

➡ ('b', 100)

2. 자료형

2-5

딕셔너리

- "키(key): 값 (value)" 쌍을 원소로 갖는데, 중괄호 { } 안에 원소(키:값)를 쉼표(,)로 구분하여 입력한다.

{ 키1:값1, 키2:값2, 키3:값3, ... }

↑ ↑ ↑
원소 1 원소 2 원소 3

[그림 2-9] 딕셔너리 구조

- 'name'(이름)과 'age'(나이)를 키로 하는 딕셔너리를 만든다. 'name' 키는 문자열 데이터 'Jay'와 짝을 이루고, 'age' 키는 숫자형 데이터 20과 짝이 된다.

〈소스〉 2.6_data_types_5.ipynb

```
[1] dict1 = {'name':'Jay', 'age':20}  
dict1
```

```
→ {'age':20, 'name':'Jay'}
```

- 키를 사용하면 매칭되는 값을 추출할 수 있다. 대괄호 [] 안에 'name' 키를 입력하면 매칭되어 있는 문자열 데이터인 'Jay'를 추출한다.

```
[3] dict1['name']
```

```
→ 'Jay'
```

2. 자료형

2-5

딕셔너리

- 대괄호 [] 안에 'age' 키를 입력하여 짝을 이루고 있는 숫자형 데이터인 20을 추출한다.

```
[4] dict1['age']
```

```
➡ 20
```

- 딕셔너리에 원소를 추가하려면 추가하려는 원소의 키를 대괄호 [] 안에 입력하고, 짝이 되는 값을 할당 연산자(=) 오른쪽에 입력한다.
- 다음은 숫자 4개를 원소로 갖는 리스트 객체를 'grade'라는 키와 매칭하여 딕셔너리(dict1)에 추가 하는 코드이다.

```
[5] dict1['grade'] = [3.0, 4.0, 3.5, 4.2]
```

```
dict1
```

```
➡ {'age':20, 'grade':[3.0, 4.0, 3.5, 4.2], 'name':'Jay'}
```

- 딕셔너리의 키를 따로 추출한다. 딕셔너리 객체에 도트 연산자(.)를 사용하여 keys() 메소드를 적용한다.

```
[6] dict1.keys()
```

```
➡ dict_keys(['name', 'age', 'grade'])
```

- 딕셔너리의 값을 따로 추출하기 위해 values() 메소드를 사용한다.

```
[7] dict1.values()
```

```
➡ dict_values(['Jay', 20, [3.0, 4.0, 3.5, 4.2]])
```

- items() 메소드를 사용하면, (키, 값) 형태의 튜플 구조로 키와 매칭되는 값을 함께 추출한다.

```
[8] dict1.items()
```

```
➡ dict_items([('name', 'Jay'), ('age', 20), ('grade', [3.0, 4.0, 3.5, 4.2])])
```

3. 연산자

3-1

산술연산자

- 숫자를 더하는 덧셈 연산자(+)를 사용하여 코드 셀에 1+2를 입력하고 실행해 본다.

<소스> 2.7_operators_1.ipynb

[1] 1+2

3

- 숫자가 아닌 문자열에 덧셈 연산자(+)를 적용하면, 문자열을 하나로 결합한다.

[2] new_str = 'Good' + ' ' + 'Morning'

new_str

'Good Morning'

[3] 2-3

-1

[4] 3*4

12

[5] 'a' * 3

'aaa'

[6] 5/3

1.6666666666666667

[7] 5//3

1

[8] 5%3

2

[9] 3**2

9

3. 연산자

3-2

논리연산자

- 코드 셀에 참을 나타내는 True를 입력해 실행한다. 논리식이 True이므로 True를 반환한다.

<소스> 2.8_operators_2.ipynb

```
[1] True
```

```
True
```

- not 연산자를 사용하여 True 값을 부정하면, 논리적으로 거짓을 나타내는 False가 된다.

```
[2] not True
```

```
False
```

명제 P	명제 Q	논리곱 ($P \wedge Q$) 또는 (P and Q)	논리합 ($P \vee Q$) 또는 (P or Q)
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

[표 2-1] 진리표

- 두 개의 명제를 and 연산자(P 그리고 Q)로 결합하는 경우, 두 명제를 모두 만족하는 경우만 참이 된다.

```
[4] print(True and True)
```

```
print(True and False)
```

```
print(False and False)
```

```
True
```

```
False
```

```
False
```

- 두 개의 명제를 or 연산자(P 또는 Q)로 결합하는 경우, 두 명제 중 하나만 만족해도 참이다.

```
[5] print(True or True)
```

```
print(True or False)
```

```
print(False or False)
```

```
True
```

```
True
```

```
False
```

3. 연산자

3-3

비교연산자

- '==' 연산자는 '서로 같다'라는 뜻을 나타낸다.

〈소스〉 2.9_operators_3.ipynb

```
[1] 3==3
```

```
True
```

- '!=' 연산자는 '서로 다르다'라는 뜻이다.

```
[2] 3!=3
```

```
False
```

- 두 변수가 갖는 값을 비교할 수 있다. var1 변수와 var2 변수가 저장하고 있는 데이터는 "a" 문자열로 동일하기 때문에 두 변수가 같다는 명제는 True 로 판별된다.

```
[5] var1='a'  
var2='a'  
var1==var2
```

```
True
```

- 숫자 값의 크기를 비교하는 부등식 연산도 가능하다.

```
[6] print(4 < 5)  
print(4 <= 5)  
print(4 >= 5)  
print(4 > 5)
```

```
True  
True  
False  
False
```

- and 논리연산자(P 그리고 Q)를 적용할 때 앞 뒤 조건식에 비교연산자를 사용하는 경우이다. 앞 뒤 조건식 모두 True 이기 때문에 and 연산자는 True 를 반환한다.

```
[7] 4 < 5 and 4 <= 5
```

```
True
```

- or 논리연산자(P 또는 Q)의 앞에 있는 조건식은 True 이고, 뒤의 조건식은 False이다. or 연산자는 두 조건식 중에서 하나만 True 이면 전체 논리식이 True 가 된다.

```
[8] 4 < 5 or 4 > 5
```

```
True
```


4. 제어문

4-1

조건문(if)

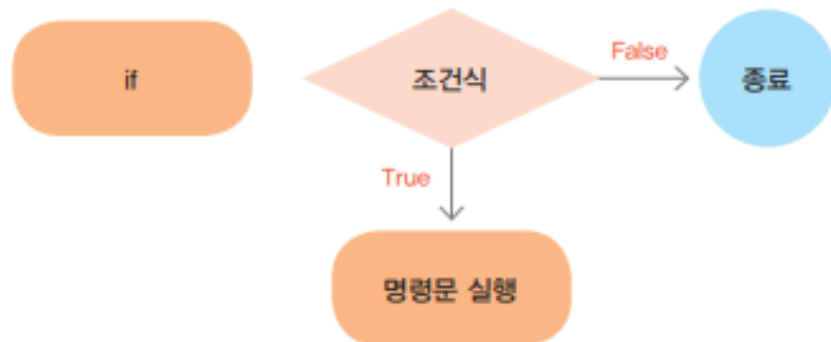
조건문은 조건식을 만족하는 경우와 그렇지 못한 경우를 구분하여 각각 다른 프로그래밍 코드를 실행한다.

일반적으로 비교연산자와 논리연산자를 조합해서 조건식을 만든다.

4-1-1

조건문을 만족할 경우에만 명령 실행(if~)

- if 조건문이 하나만 있는 경우 조건식을 만족하면 해당 명령문을 실행하고, 그렇지 않은 경우에는 if 문을 종료하고 그 다음의 코드를 실행하게 된다.



[그림 2-10] if 조건문

<소스> 2.10_control_1.ipynb

```
[1] a = 4
    if a % 2==0:
        print("a는 짝수")
        print("a는 2의 배수")
```

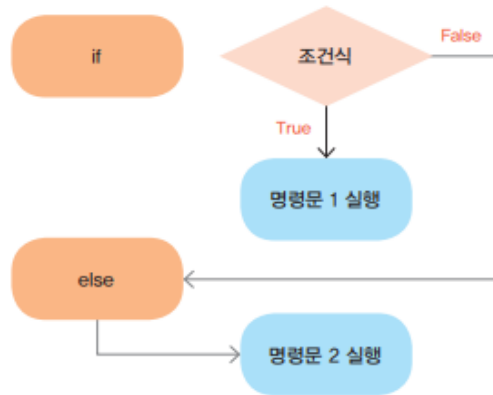
```
a는 짝수
a는 2의 배수
```

Tip if 조건문을 만족할 때 실행되는 명령문 코드는 들여쓰기를 해준다. 일반적으로 4칸 들여쓰기를 한다. 들여쓰기 상태에서 여러 줄로 입력해도 된다.

4. 제어문

4-1-2 조건문을 만족하는 경우와 만족하지 않는 경우(if~else)

- if~else 문을 사용하면, 조건문을 만족하는 경우와 그렇지 않은 경우를 구분하여 프로그램을 제어할 수 있다.

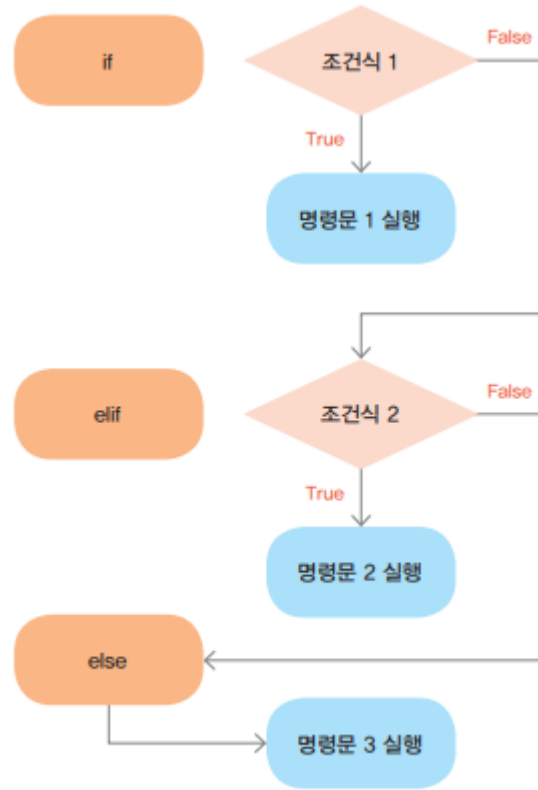


[그림 2-11] if~else 조건문

```
[3] b = 7
    if b % 2 == 0:
        print("b는 짝수")
    else:
        print("b는 홀수")
```

➡ b는 홀수

4-1-2 여러 조건을 중첩하여 적용(if~elif~else)



[그림 2-12] if~elif~else 조건문

- if 문과 함께 elif 문을 여러 개 사용하면, 서로 다른 조건식을 여러 개 중첩하여 적용할 수 있다.

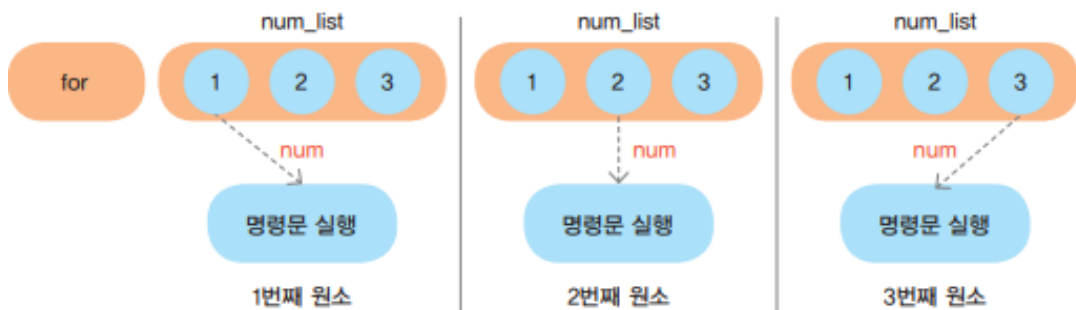
4. 제어문

4-2

for 반복문

- 반복문은 동일한 명령을 여러 번 반복적으로 처리하고 싶을 때 사용한다.

- for 반복문을 사용하여 숫자 1, 2, 3을 순서대로 출력한다. 숫자 1, 2, 3이 원소인 리스트 (num_list)의 첫 번째 원소인 1이 num 변수에 할당되어 print 명령으로 실행된다. 그리고 두 번째 원소 2가 num 변수에 할당되어 출력되고, 마지막으로 원소인 3이 출력 된다.



[그림 2-13] for 반복문

<소스> 2.11_control_2.ipynb

```
[1] num_list = [1, 2, 3]
    for num in num_list:
        print(num)
```

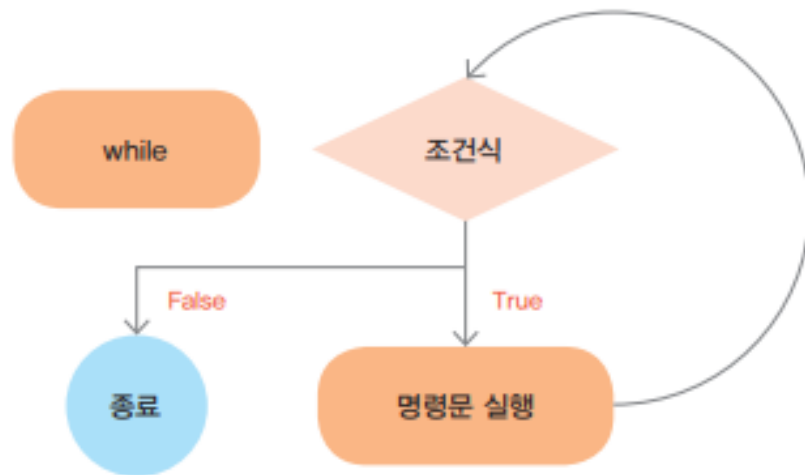
```
➡ 1
   2
   3
```

4. 제어문

4-3

while 반복문

- for 반복문의 반복 횟수는 반복할 객체의 원소 개수만큼 한정된다.
- 반면, while 반복문은 조건식을 만족하는 True 인 경우에는 명령문을 실행하고 다시 조건식을 비교하는 과정을 무한 반복한다.
- 다만 조건식을 만족하지 못하는 False 일 때는 반복문이 종료된다.



[그림 2-14] while 반복문

<소스> 2.12_control_3.ipynb

```
[1] num = 1
    while num < 4:
        print(num)
        num = num + 1
```

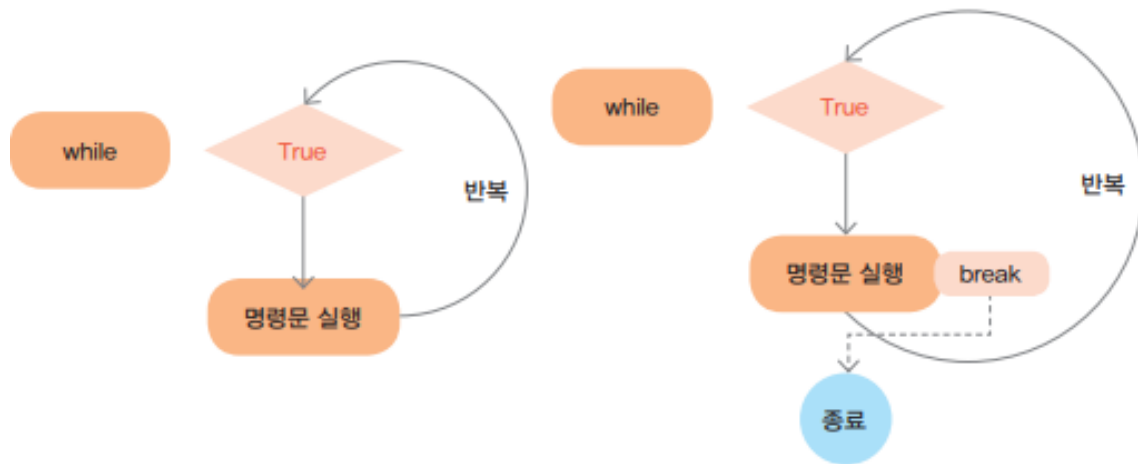
```
1
2
3
```

4. 제어문

4-3-1

break 명령

- while 반복문에서 조건식 위치에 True 를 입력하면 조건식이 항상 참이므로 들여쓰기 된 명령문 들은 무한 반복하며 실행된다.
- 이런 경우 반복문을 빠져나오기 위해 break 명령문을 사용한다.



[그림 2-15] break 명령

```
[3] num = 1
    double = []

    while True:
        double.append(num * 2)
        if len(double) == 3:
            break
        num += 1

    print(double)
```

➡ [2, 4, 6]

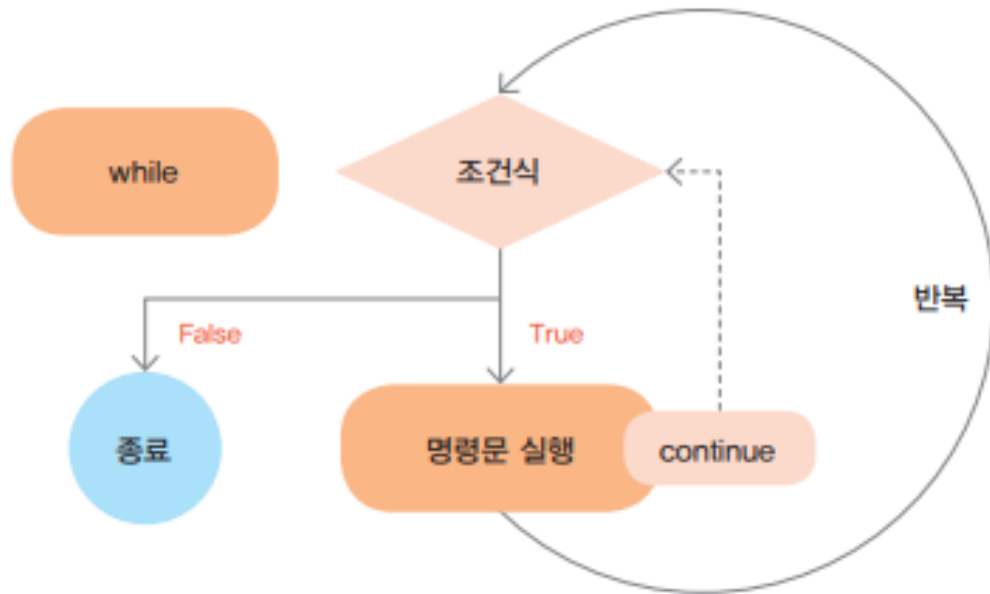
Tip len 함수는 리스트 원소의 개수를 계산한다.

4. 제어문

4-3-2

continue 명령

- continue 명령은 while 반복문에서 continue 명령 뒤에 오는 나머지 코드를 실행하지 않고 조건식을 판별하는 while 문의 처음으로 돌아가게 한다.



[그림 2-16] continue 명령

```
[6] lotto = []
while len(lotto) < 6:
    random.shuffle(num_list)
    num_selected = num_list[0]
    if num_selected in lotto:
        continue
    lotto.append(num_selected)
    print(num_selected)

print(lotto)
```

```
➡ 35
   18
   25
   43
   45
   11
   [35, 18, 25, 43, 45, 11]
```

4. 제어문

4-4

예외처리 (try~except)

- 오류가 발생하면 프로그램이 정상적으로 실행되지 않기 때문에 예외처리 구문을 활용해서 부분적으로 오류가 발생하더라도 나머지 코드를 실행하도록 설계할 때가 있다.
- try~except 구문은 어떤 코드를 실행(try)해 보고, 오류가 발생하면 예외적으로 (except) 준비된 코드를 실행하는 방식으로 예외 처리를 한다.
- 따라서 try 부분 또는 except 부분 중에서 어느 하나만 실행된다. try 부분에 오류가 발생하면 except 부분이 무조건 실행된다.
- 다음의 예제는 딕셔너리에 'address'라는 키가 존재하면 매칭된 값을 출력하고 (try~부분), 오류가 발생하면 "주소 정보가 없습니다"라는 메시지를 출력한다 (except~부분).

```
[2] data = {'name':'Jay', 'age':20, 'grade':[3.0, 4.0, 3.5, 4.2]}

try:
    print("주소", data['address'])
except:
    print("주소 정보가 없습니다")
```

➡ 주소 정보가 없습니다

- 오류 발생 여부와 관계없이 반드시 실행해야 하는 코드가 있을 때는 finally 구문을 추가한다. 따라서 try 부분과 finally 부분이 실행되거나 except 부분과 finally 부분이 실행된다.
- 다음의 예제에서는 'name' 키를 사용하여 데이터를 추출하는데 오류가 없기 때문에 try 부분이 실행되고 except 부분이 실행되지 않는다. 추가적으로 finally 부분이 실행된다.

```
[3] data = {'name':'Jay', 'age':20, 'grade':[3.0, 4.0, 3.5, 4.2]}

try:
    print("이름", data['name'])
except:
    print("이름 정보가 없습니다")
finally:
    print("모든 작업이 완료되었습니다")
```

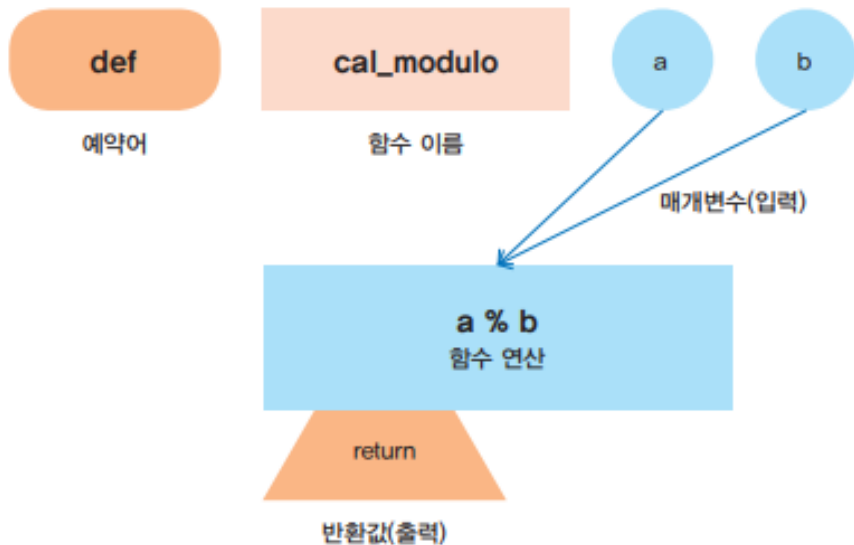
➡ 이름:Jay
모든 작업이 완료되었습니다

5. 함수

5-1

사용자 정의 함수

- 파이썬 함수는 예약어 `def`를 사용하여 정의한다.
- 예제에서 `cal_modulo`는 함수의 이름이고 괄호 () 안의 `a`와 `b`는 입력값을 나타내는 매개변수다. 콜론(:)을 끝부분에 입력한다.
- 콜론 다음 줄에 4칸 들여쓰기로 시작되는 코드 블록이 함수의 연산에 해당하는 부분이다. `a`를 `b`로 나눈 나머지를 `temp`에 저장하고, `return` 명령으로 반환하는 `temp`가 출력값(`y`)에 해당한다.



[그림 2-17] 사용자 정의 함수

〈소스〉 2.14_function_1.ipynb

```
[1] def cal_modulo(a, b):  
    temp = a % b  
    return temp  
    cal_modulo(5, 3)
```

➡ 2

5. 함수

- 다음과 같이 함수가 반환하는 출력값을 변수에 저장할 수 있다.
cal_modulo(10, 3) 코드는 10을 3으로 나눈 나머지 1을 반환하는데 이 값을 modulo 변수에 할당한다.

```
[2] modulo = cal_modulo(10, 3)
    print(modulo)
```

➡ 1

- (a, b)와 같은 형태로 2개의 숫자를 원소로 갖는 튜플을 사용하면, cal_modulo 함수의 매개변수 인 a, b에 입력값으로 전달할 수 있다.
- 다음의 num_pairs 리스트는 3개의 튜플을 원소로 갖는다. for 반복문을 이용하여 각 튜플의 a, b 위치의 숫자를 cal_modulo(a, b) 함수의 입력값으로 전달하고 그 결과를 print 명령으로 출력한다.

```
[3] num_pairs = [(5, 3), (2, 2), (10, 3)]
    for a, b in num_pairs:
        modulo = cal_modulo(a, b)
        print(modulo)
```

➡ 2

0

1

- 다음은 2개의 숫자 쌍을 원소로 갖는 리스트(num_pairs)를 입력받아, 각 숫자 쌍에 대한 나머지 연산의 결과를 딕셔너리 형태로 정리하는 cal_pairs_modulo 함수이다.

```
[4] def cal_pairs_modulo(num_pair_list):
    result = {}
    for a, b in num_pairs:
        modulo = a % b
        result[(a,b)] = modulo
    return result
```

```
mod_pairs = cal_pairs_modulo(num_pairs)
mod_pairs
```

➡ {(2, 2):0, (5, 3):2, (10, 3):1}

- 함수의 연산을 처리하는 코드 블록에 다른 함수를 불러와 사용할 수 있다. 앞에서 정의한 cal_pairs_modulo 함수를 사용하여 함수의 결과값을 출력하는 print_mod_pairs 함수를 정의한다.

```
[5] def print_mod_pairs():
    print(cal_pairs_modulo(num_pairs))

    print_mod_pairs()
```

➡ {(5, 3):2, (2, 2):0, (10, 3):1}


5. 함수

5-2

람다(lambda) 함수

- 람다 함수를 사용하는 이유는 함수를 간단한 표현으로 정의할 수 있기 때문이다. lambda 입력 값(x):출력값(y) 형태로 정의한다.
- lambda x: add_one_lambda.append(x+1)는 x를 입력받아 숫자 1을 더하고 그 값을 리스트(add_one_lambda)에 추가하는 람다 함수이다. 리스트 [1, 2, 3]의 원소를 하나씩 람다 함수에 입력하고 add_one_lambda 리스트를 출력한다.

```
[3] add_one_lambda = []  
    add_func = lambda x:add_one_lambda.append(x+1)  
    for x in [1, 2, 3]:  
        add_func(x)  
  
    print(add_one_lambda)
```

 [2, 3, 4]

- 두 개의 입력값 x, y를 받아 덧셈을 하는 람다 함수를 정의한다. 매개변수 x에 2가 입력되고, y는 3이 대입된다.

```
[4] new_add_func = lambda x, y:x+y  
    answer = new_add_func(2, 3)  
    print(answer)
```

 5

5. 함수

5-3

파이썬 내장 함수

5-3-1

sum 함수

- 원소들의 합계를 구한다. 1~10까지 숫자를 원소로 갖는 리스트(numbers)를 매개변수의 값으로 전달하면 리스트 원소들의 합계를 계산한다.

〈소스〉 2.16_function_3.ipynb

```
[1] numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
    sum(numbers)
```

➡ 55

5-3-2

max 함수

- 원소 중에서 최대값을 찾는다.

```
[2] max(numbers)
```

➡ 10

5-3-3

min 함수

- 원소 중에서 최소값을 찾는다.

```
[3] min(numbers)
```

➡ 1

5. 함수

5-3-4

len 함수

- 원소의 개수 또는 문자열의 길이를 계산한다.

```
[4] len(numbers)
```

```
10
```

5-3-5

enumerate 함수

- 리스트, 튜플 등 원소값에 순서(인덱스)가 있는 자료형을 입력받아 개별 원소를 인덱스 숫자와 함께 반환해 준다.
- 다음 예제에서, i는 인덱스를 나타내고 num은 원소값을 나타낸다.

```
[5] for i, num in enumerate(numbers):
```

```
    print(i, num)
```

```
0 1
```

```
1 2
```

```
2 3
```

```
3 4
```

```
4 5
```

```
5 6
```

```
6 7
```

```
7 8
```

```
8 9
```

```
9 10
```

Tip for 반복문에서 i, num과 같은 변수 이름은 사용자가 자유롭게 지정할 수 있다. 예를 들어, i 대신 idx, num 대신에 n을 사용할 수 있다. if나 while 등 예약어를 사용할 수는 없다.

5. 함수

5-3-6

range 함수

- 입력값 범위에 해당하는 정수의 배열을 만든다. range(10)과 같이 숫자를 1개 입력하면 0부터 9까지 범위에 해당하는 range 객체를 만든다.
- range 함수는 입력값으로 2개의 값을 전달하면 범위의 시작과 끝을 의미한다. range(1, 10)은 1부터 시작하고 9로 끝나는(범위의 끝 값인 10을 포함하지 않는) range 객체를 만든다.

```
[6] range(10)
```

```
→ range(0, 10)
```

```
[7] list(range(1, 10))
```

```
→ [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

5-3-7

list 함수

- 입력받은 데이터를 리스트로 변환한다. range 배열을 리스트로 변환해 본다.

```
[8] print(list(range(10)))
```

```
→ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

5. 함수

5-3-8

eval 함수

- 문자열을 입력받아 파이썬 코드로 변환하여 실행한다.

```
[9] eval('print(numbers)')
```

```
➡ [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

5-3-9

map 함수

- 여러 개의 원소를 갖는 자료형(리스트, 튜플 등)과 이들 원소를 입력값으로 받는 함수를 매개변수로 갖는다.
- 예제에서 map 함수는 add_one 함수와 numbers(리스트)를 입력받는다. numbers 리스트의 원소들은 하나씩 add_one 함수에 입력되고 함수를 실행한 결과값(1을 더한 값)이 순서대로 반환된다.

```
[10] add_one = lambda x:x+1
```

```
results = map(add_one, numbers)
```

```
print(list(results))
```

```
➡ [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

5. 함수

5-3-10

filter 함수

- map 함수와 비슷하게 반복 가능한 여러 원소를 갖는 자료형과 각 원소를 입력받을 수 있는 함수를 매개변수로 입력받는다.
- 예제에서 numbers 리스트의 숫자들은 하나씩 even_num 함수에 입력되고, even_num의 출력값이 True 일 때만 filter 함수의 출력값에 포함된다. 따라서 2로 나눈 나머지가 0이 되는 수(짝수)들만 results 변수에 저장된다.

```
[11] even_num = lambda x:x%2==0
      results = filter(even_num, numbers)

      print(list(results))
```

 [2, 4, 6, 8, 10]

5-3-11

int 함수

- 실수를 정수형으로 변환한다.

```
[12] int(3.14)
```

 3

5. 함수

5-3-12

str 함수

- 입력값을 문자열 자료형으로 변환한다.

```
[13] str(3.14)
```

```
➡ '3.14'
```

5-3-13

round 함수

- 숫자를 입력받아 반올림을 한다.

```
[14] round(3.14)
```

```
➡ 3
```

5-3-14

reversed 함수

- 원소들이 위치하는 순서를 정반대로 뒤집어 준다.

```
[15] numbers_reversed = reversed(numbers)
```

```
print(list(numbers_reversed))
```

```
➡ [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```


5. 함수

5-3-15

sorted 함수

- 배열의 순서를 오름차순으로 정렬한다.

```
[16] sorted([3, 2, 1])
```

```
➡ [1, 2, 3]
```

5-3-16

zip 함수

- 원소 개수가 같은 자료형들에 대하여 각 원소들을 인덱스 순서대로 매핑하여 짝을 짓는다.

```
[17] chars = ['a', 'b', 'c']
```

```
nums = [1, 2, 3]
```

```
pairs = zip(chars, nums)
```

```
print(list(pairs))
```

```
➡ [('a', 1), ('b', 2), ('c', 3)]
```

6. 클래스

클래스는 여러 개의 함수를 내부 메소드(method)로 지정할 수 있고, 다양한 자료형 데이터를 내부 속성으로 가질 수 있다. 다양한 속성과 함수 기능을 정의해 두면 동일한 속성과 기능을 갖는 클래스 객체를 여러 개 만들어 사용할 수 있다. 프로그램 코드가 간결해지는 장점이 있다.

- 숫자 두 개를 더하는 계산기 기능을 하는 클래스를 정의한다. 예약어 `class`를 사용한다.
- 예제는 `Calculator`라는 이름을 사용하고 클래스 이름 뒤에 콜론(:)을 붙인다. 클래스 내부에는 들여쓰기 블록에 2개의 함수(`__init__`, `add`)를 정의한다. `__init__` 함수는 클래스 객체가 생성될 때 자동 실행된다. `add` 함수는 외부에서 메소드로 호출될 때 실행된다.

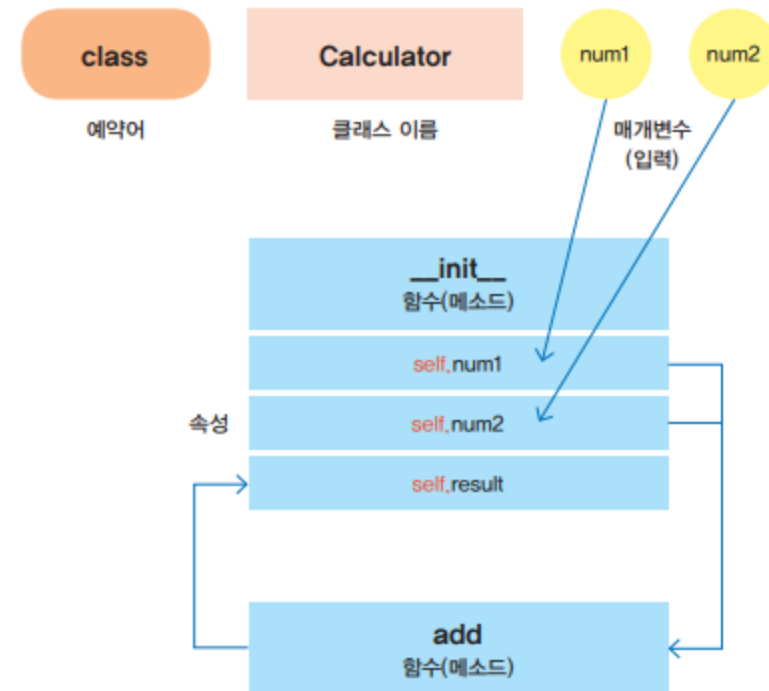
```
[2] class Calculator:

    def __init__(self, num1, num2):
        self.num1 = num1
        self.num2 = num2
        self.result = 0

    def add(self):
        self.result = self.num1 + self.num2
        return self.result

cal = Calculator(1, 2)
print(cal.num1, cal.num2, cal.result)
```

➡ 1 2 0



- 앞에서 `Calculator(1, 2)`와 같이 클래스를 호출하면 `__init__` 함수가 실행되고, `num1` 위치의 1이 `self.num1` 속성에 할당되고, `num2` 위치의 값인 2가 `self.num2` 속성에 할당된다.
- `self`는 클래스 객체 자신을 가리킨다. `cal` 클래스 객체의 `num1` 속성(`cal.num1`)은 1이고 `num2` 속성(`cal.num2`)은 2가 된다

6. 클래스

- cal 클래스 객체의 add 함수(클래스 메소드)는 self.num1와 self.num2를 더한 결과를 self.result 변수에 저장하고, 그 값을 출력값으로 반환한다.
- 따라서 cal.add()와 같이 add 메소드를 실행하면 1과 2를 더한 값인 3이 self.result 속성에 저장되고 이 값을 출력으로 반환한다.

```
[3] cal.add()
```

```
→ 3
```

- cal.result와 같이 cal 클래스 객체의 result 속성값을 확인한다. result 속성에 1과 2를 더한 3이 저장되어 있다.

```
[4] cal.result
```

```
→ 3
```

- Calculator 클래스 정의를 바꿔본다. 두 개의 숫자를 입력받아 사칙연산(덧셈, 뺄셈, 곱셈, 나눗셈)을 모두 처리할 수 있는 메소드를 각각 정의한다.
- 그리고 change 메소드는 클래스 객체가 가지고 있는 num1, num2 속성값을 변경하는 함수로 정의한다.

```
[5] class Calculator:

    def __init__(self, num1, num2):
        self.num1 = num1
        self.num2 = num2
        self.result = 0

    def add(self):
        self.result = self.num1 + self.num2
        return self.result

    def subtract(self):
        self.result = self.num1 - self.num2
        return self.result

    def multiply(self):
        self.result = self.num1 * self.num2
        return self.result

    def divide(self):
        self.result = self.num1 / self.num2
        return self.result

    def change(self, num1, num2):
        self.num1 = num1
        self.num2 = num2
        print("num1:", self.num1)
        print("num2:", self.num2)
```

6. 클래스

- Calculator(1, 2)와 같이 cal2 클래스 객체를 만들고, num1 속성과 num2 속성을 출력 하면 1과 2라는 것이 확인된다.

```
cal2 = Calculator(1, 2)
print(cal2.num1, cal2.num2)
```

```
➡ 1 2
```

- 클래스 메소드를 실행해 본다. cal2 객체에 사칙연산 메소드를 각각 적용한다. 각 연산의 결과를 print 함수로 출력한다.

```
[6] print(cal2.add())
    print(cal2.subtract())
    print(cal2.multiply())
    print(cal2.divide())
```

```
➡ 3
   -1
   2
   0.5
```

- cal2 클래스 객체의 change 메소드를 적용한다. self.num1, self.num2 위치에 순서대로 3과 2를 입력한다. self.num1의 값이 3으로 변경되고, self.num2에는 2가 입력된다.

```
[7] cal2.change(3, 2)
```

```
➡ num1:3
   num2:2
```

- 변경된 num1, num2 속성값으로 add 메소드가 잘 실행되는지 확인한다. 3과 2를 더한 값인 5가 출력값으로 반환되는 것을 알 수 있다.

```
[8] cal2.add()
```

```
➡ 5
```

7. 실습

7-1

나만의 계산기 만들기

- 입력으로 두 개의 숫자를 받아 계산하는 기능을 갖춘 계산기를 만들어 보자. (덧셈, 뺄셈, 곱셈, 나눗셈 포함)

1. 계산기 함수 설계

```
1 def add(x, y):  
2     return x + y  
3  
4 def subtract(x, y):  
5     return x - y  
6  
7 def multiply(x, y):  
8     return x * y  
9  
10 def divide(x, y):  
11     return x / y
```

2. 사용자 입력 처리

```
1 def get_user_input():  
2     num1 = float(input("첫 번째 숫자를 입력하세요: "))  
3     operator = input("연산자를 입력하세요 (+, -, *, /): ")  
4     num2 = float(input("두 번째 숫자를 입력하세요: "))  
5     return num1, operator, num2
```

3. 계산 함수 실행

```
1 def calculate(num1, operator, num2):  
2     if operator == "+":  
3         return add(num1, num2)  
4     elif operator == "-":  
5         return subtract(num1, num2)  
6     elif operator == "*":  
7         return multiply(num1, num2)  
8     elif operator == "/":  
9         return divide(num1, num2)  
10    else:  
11        return "잘못된 연산자입니다."
```

4. 메인 함수와 실행

```
1 def main():  
2     num1, operator, num2 = get_user_input()  
3     result = calculate(num1, operator, num2)  
4     print(f"결과: {result}")  
5  
6 if __name__ == "__main__":  
7     main()
```

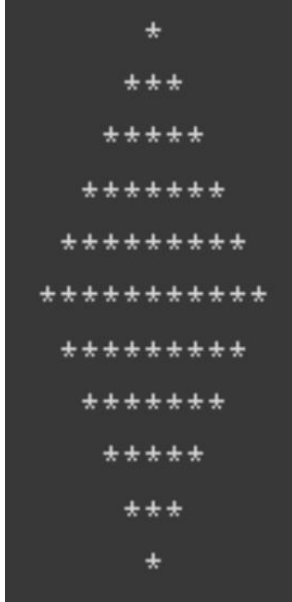
7. 실습

7-2

'*' 별 찍기 게임

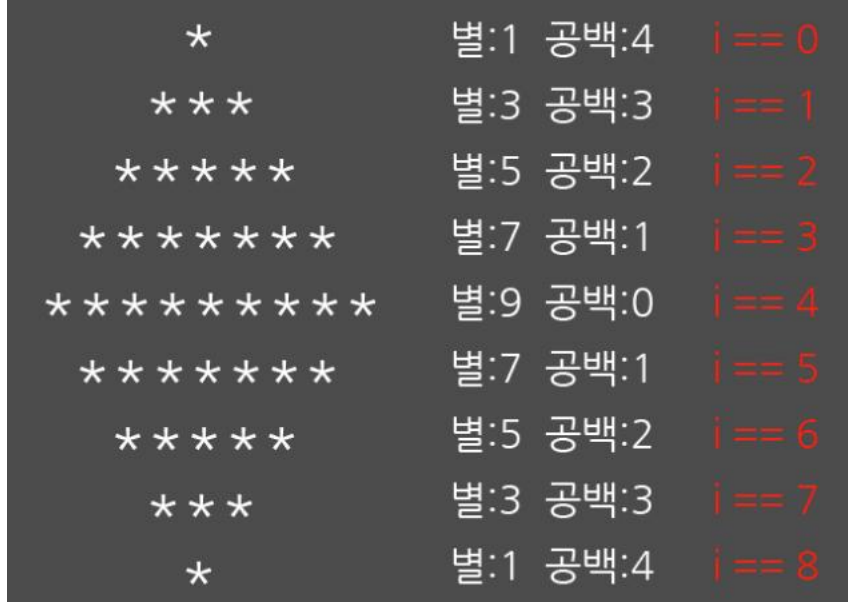
- 사용자가 숫자를 넣으면 해당 층의 개수를 갖는 '*' 로 이루어진 다이아몬드를 그려주는 게임을 만들어 보자.

1. 게임 예시



n=11

2. 예시 설명



3. 예제 코드

```
a = 11
for i in range(a//2):
    print(' ' * (a//2 - i), end = '')
    print('*' * (2*i+1))

for i in range(a//2-1):
    print(' ' * (i + 2), end = '')
    print('*' * ((a//2*2)-3-2*i))
```