
python

❖ 학습 내용

- Python 자료구조
- Python 제어문

❖ 자료 구조

◆ 정의

- **자료구조**(Data Structure)는
  컴퓨터 과학에서 효율적인 접근 및 수정을 가능케 하는 자료의 조직, 관리, 저장을 의미한다.
- 다수의 데이터를 편리하게 다룰 수 있게 도와 주는 자료형

◆ 종류

- 리스트(List) 자료형
  - 데이터의 목록을 다루는 자료형
  - Sequence 자료형
  - 인덱스를 통해 데이터에 접근
  - 데이터의 변경 가능 (Mutable)

- 튜플(Tuple) 자료형
  - 사전적 정의 : 몇개의 요소로 된 집합
  - **변경 불가능한 자료구조 (Immutable)**

- 딕셔너리(Dictionary) 자료형
  - 사전 이라고 부르기도 한다.
  - Key와 Value를 한 쌍으로 갖는 자료형
  - Key를 통해서 Value를 얻는다.

- 집합(Set) 자료형
  - 집합에 관련된 자료형
  - 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형

❖ 리스트 (List)

- List는 변경 가능한 자료형이다.(Mutable Type)

◆ 리스트의 형식

```
[ 데이터1, 데이터2, 데이터3, . . . , 데이터n ]
```

◆ 리스트의 정의와 기본 연산

- 리스트: 임의의 객체를 순차적으로 저장하는 집합적 자료형
- 문자열이 지닌 대부분의 연산들은 리스트도 지원

❖ 리스트 (List)

◆ 리스트의 정의와 기본 연산

➡ ex 1]

```
L = [1,2,3]
print(type(L))
print(len(L))
print(L[1])
print(L[-1])
print(L[1:3])
print(L + L)
print(L * 3)
```

```
<class 'list'>
3
2
3
[2, 3]
[1, 2, 3, 1, 2, 3]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

➡ 설 명 1]

- 리스트: 임의의 객체를 순차적으로 저장하는 집합적 자료형
- 리스트는 객체를 순차적으로 저장하므로 인덱스가 존재
- L=[1, 2, 3] → 1은 인덱스 0, 2는 인덱스 1, 3은 인덱스 2
- type(L) → L의 타입은 무엇인가?
- L에 할당된 개체를 통해 list로 판단
- len(L) → L의 길이(원소의 갯수)는 얼마인가?
- L[1] → 인덱스가 1인 값
- L[-1] → 인덱스가 뒤에서 첫 번째인 값
- L[1:3] → 인덱스 1(2)부터 인덱스 3 앞(3)까지 슬라이싱
- 리스트 + 리스트 → 두 리스트의 원소를 합쳐 하나의 리스트로
- 리스트 * 숫자 → 리스트를 숫자만큼 반복하여 하나의 리스트로
- +(더하기), *(곱하기), 슬라이싱, 인덱싱은 문자열과 동일하게 행동

❖ 리스트 (List)

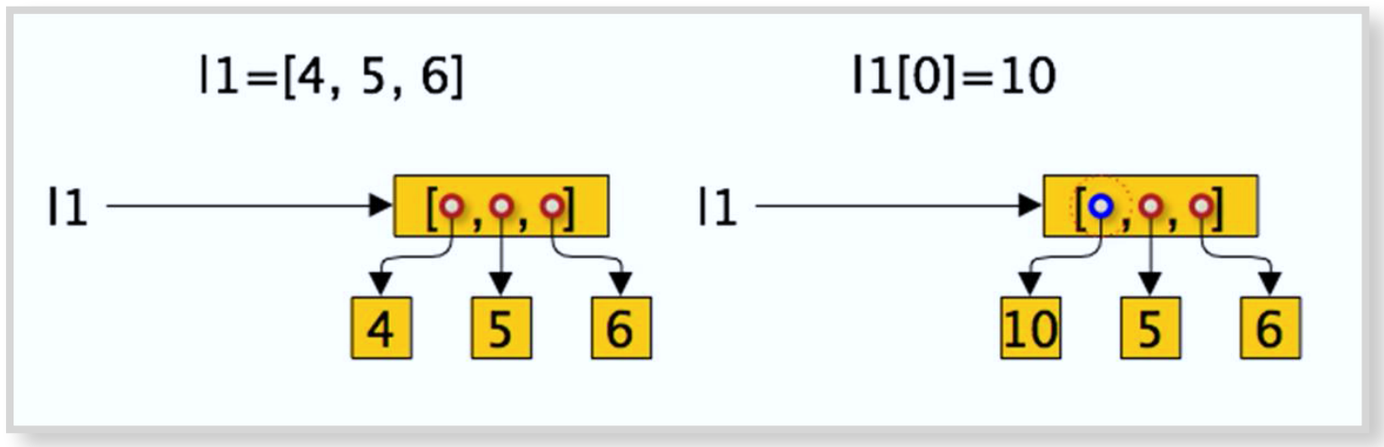
◆ 리스트의 정의와 기본 연산

➡ ex 2]

```
l1 = [4,5,6]
l1[0] = 10
print(l1)
```

```
[10, 5, 6]
```

➡ 설 명 2]



- 리스트 → 변경 가능 (문자열 → 변경 불가능, 에러 발생)
- l1[0] = 10 → l1의 인덱스 0(현재 4)를 10으로 하겠다는 의미
- l1은 4, 5, 6을 담고 있는 리스트를 가리키는 레퍼런스를 가짐
- 1[0] = 10 → 기존의 4는 쓰레기가 되고 새로운 레퍼런스로 수정

❖ 리스트 (List)

◆ 리스트의 정의와 기본 연산

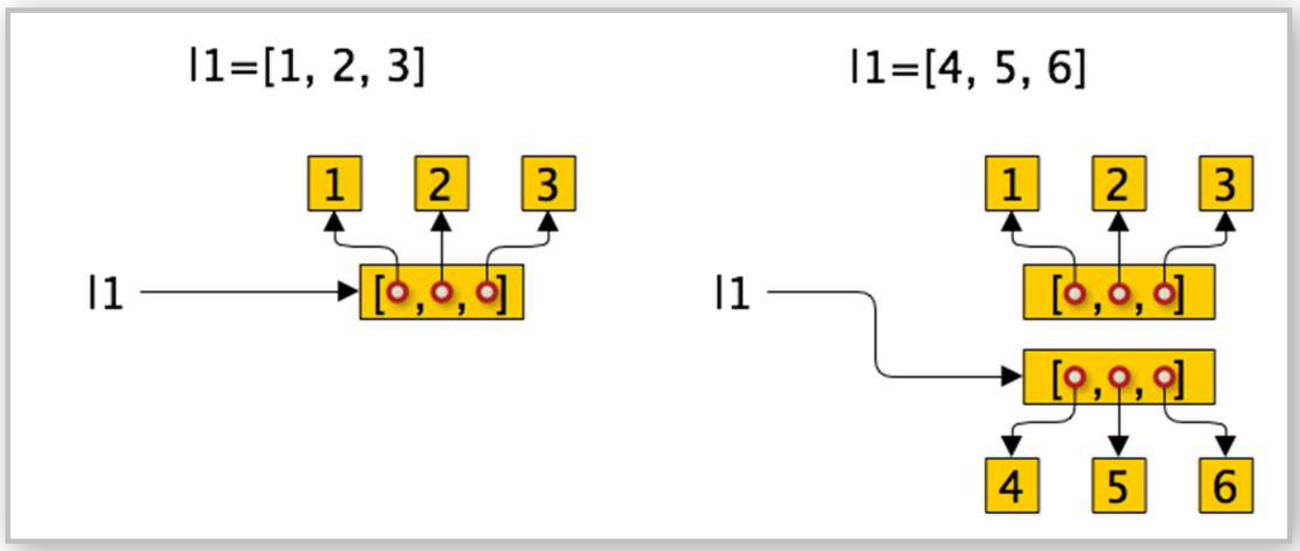
- 동일한 변수에 다른 리스트를 할당하는 것은 해당 변수의 레퍼런스를 변경함

➡ ex 3]

```
l1 = [1,2,3]
l1 = [4,5,6]
print(l1)
```

```
[4, 5, 6]
```

➡ 설 명 3]



- l1을 [1, 2, 3] 리스트에서 [4, 5, 6] 리스트로 다시 할당
- l1 = [4, 5, 6] → 기존의 리스트는 쓰레기, 새로운 레퍼런스로 수정

❖ 리스트 (List)

◆ range() 함수를 통한 인덱스 리스트 생성

- range(k): 0부터 k-1까지의 숫자의 리스트를 반환함
- list() : 여러개의 데이터를 리스트로 반환함

➡ ex 1]

```
L = list(range(10))
print(L)
print(L[::2])
print(L[::-1])
print(4 in L )
```

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

➡ 설 명 1]

- range(k) → [0 , 1, …, k-1]로 리스트 구성하여 반환
- range(10) → 0부터 10-1인 9까지 있는 리스트 반환
- L[::2] → 전체 리스트에서 인덱스가 2씩 차이나게 슬라이싱
- L[::-1] → 전체 리스트를 거꾸로 슬라이싱
- 4 in L (멤버쉽 테스트) → L 안에 4가 있다(true)

❖ 튜플 (Tuple)

- **Tuple은 변경 불가능한 자료형이다.(Imutable Type)**

◆ 튜플의 형식

(데이터1, 데이터2, 데이터3, . . . , 데이터n)

◆ 튜플의 정의와 기본 연산

- 튜플 : 리스트와 유사하지만 **튜플 내의 값을 변경할 수 없음(Immutable)**
- 적합한 사용 예
  - months = ('January','February','March','April','May','June','July',
    'August','September', 'October','November',' December')
  - 각 값에 대해 인덱스가 부여됨
- 문자열이 지닌 대부분의 연산들은 튜플도 지원

- 튜플: 리스트와 유사하지만 값 변경 불가능
- 튜플은 둥근 괄호 I 소괄호 ()를 사용
- months라는 튜플에 January부터 December까지 있음
- 튜플도 인덱스가 존재, January는 인덱스 0, December는 인덱스 11
- months의 길이는 12, 마지막 인덱스(December)는 11
- 튜플은 리스트와 달리 수정 불가능
- 수정이 불가능하므로 상수와 비슷한 속성

❖ 튜플 (Tuple)

◆ 튜플의 정의와 기본 연산

➡ ex 1]

```
t = (1,2,3)
print(len(t))
print(t[0])
print(t[-1])
print(t[0:2])
print(t[::2])
print(t + t + t)
print(t * 3)
print(3 in t)
```

```
3
1
3
(1, 2)
(1, 3)
(1, 2, 3, 1, 2, 3, 1, 2, 3)
(1, 2, 3, 1, 2, 3, 1, 2, 3)
True
```

❖ 튜플 (Tuple)

◆ 튜플의 상수적 성격

- 튜플은 내용 변경 불가

➡ ex 1]

```
t = (1,2,3)
t[0] = 100
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

➡ 설 명 1]

- 튜플 → 상수와 같이 변경 불가능
- t[0] = 100 → t의 인덱스 0인 1을 100으로 변경(불가)

- 반면에 리스트는 내용 변경 가능

➡ ex 2]

```
L = [1,2,3]
L[0] = 100
print(L)
```

```
[100, 2, 3]
```

➡ 설 명 2]

- 리스트 → 변경 가능

❖ 사전 (Dictionary)

- 시퀀스 자료형이 아니다. 인덱스가 없음, 키를 이용해 값 저장(매핑)

◆ 사전의 형식

{ 키1 : 데이터1, 키2 : 데이터2, 키3 : 데이터3, . . . , 키n : 데이터n }

◆ 사전의 정의와 기본 사용법

- 정수형 인덱스가 아닌 키를 이용하여 값을 저장하는 자료 구조
  - 저장된 각 자료에 대한 순서는 의미 없음
- 매핑(Mapping) 함수와 비슷한 역할을 함
  - x라는 키값을 넣으면 값 y를 반환함

- 사전(dictionary)도 활용 빈도가 높은 자료구조
- 사전 → 인덱스가 없음, 키를 이용해 값 저장(매핑)

❖ 사전 (Dictionary)

◆ 사전의 정의와 기본 사용법

➡ ex 1]

```
d = {'one': 'hana', 'two': 'dul', 'three': 'set'}
print(d['one'])
```

```
hana
```

➡ 설 명 1]

- 사전 → 중괄호({}) 사용
- 콤마(,)를 기준으로 아이템 구분
- d = {키:벨류, 키:벨류, 키:벨류} → 사전 d의 원소 3개
- 사전의 원소 → 키:벨류(쌍으로 이루어짐)
- print d[‘one’] → 사전 d 중 one이라는 키의 벨류 출력

❖ 사전 (Dictionary)

◆ 사전의 정의와 기본 사용법

➜ ex 2]

```
d = {'one': 'hana', 'two': 'dul', 'three': 'set'}
d['four'] = 'net' # 새 항목의 삽입
print(d)
d['one'] = 1 # 기존 항목의 값 변경
print(d)
print('one' in d) # 키에 대한 멤버쉽 테스트
```

```
{'four': 'net', 'three': 'set', 'two': 'dul', 'one': 'hana'}
{'four': 'net', 'three': 'set', 'two': 'dul', 'one': 1}
True
```

➜ 설 명 2]

- 사전에 새 항목 넣기 → 사전['키']='벨류'
- 사전은 순차적이지 않고, 랜덤한 순서로 존재함
- 실제로는 키에 대한 해시값으로 순차 정렬이 됨
- 자세한 내용은 "사전, 해시"로 검색하여 참고
- one의 벨류를 hana에서 1로 변경하기
- d['one'] = 1 → 키 one에 매핑된 기존 벨류 hana를 1로 변경
- 'one' in d → one이라는 키가 d에 존재하는지 확인
- 사전에서 기본적인 연산자는 키값을 사용
- 'one' in d → 4개의 키값 중 one이 존재하는지 확인

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 사전 (Dictionary)

◆ 사전의 정의와 기본 사용법

➡ ex 3]

```
d = {'one': 1, 'two': 'dul', 'three': 'set', 'four': 'net'}
print(d.keys()) # 키만 리스트로 추출함
print(d.values()) # 값만 리스트로 추출함
print(d.items()) # 키와 값의 튜플을 리스트로 반환함
```

```
['four', 'three', 'two', 'one']
['net', 'set', 'dul', 1]
[('four', 'net'), ('three', 'set'), ('two', 'dul'), ('one', 1)]
```

➡ 설 명 3]

- 사전.keys( ) → 키만 추출(임의의 순서)
- 사전.values( ) → 벨류만 추출(임의의 순서)
- 사전.items( ) → 키와 값을 튜플로 추출(임의의 순서)
- Keys, values, items가 반환하고 있는 자료형 → 리스트

❖ 내장 자료형의 정리와 객체 신원 파악

- **내장 자료형 → 수치형, 문자열, 리스트, 튜플, 사전**

◆ 내장 자료형의 특성 정리

자료형	저장 / 접근 방법	변경 가능성	저장 모델
수치형	직접(Direct)	변경불가능(Immutable)	리터럴(Literal)
문자열	시퀀스(Sequence)	변경불가능(Immutable)	리터럴(Literal)
리스트	시퀀스(Sequence)	변경가능(Mutable)	컨테이너(Container)
튜플	시퀀스(Sequence)	변경불가능(Immutable)	컨테이너(Container)
사전	매핑(Mapping)	변경가능(Mutable)	컨테이너(Container)

- 내장 자료형 → 수치형, 문자열, 리스트, 튜플, 사전
- **문자열, 리스트, 튜플 → 시퀀스(인덱스 존재)**
- **사전 → 매핑(인덱스 없음)**
- **리스트, 사전 → 변경 가능 / 나머지는 불가능**
- 수치형, 문자열 → 리터럴
- 리터럴은 정수나 숫자, 표기법 등이 하나씩 저장됨
- 0x→ 16진법을 나타내는 리터럴(표기법)
- 표기법을 사용하여 숫자(수치형 자료)를 나타냄
- -0.2e-4도 하나의 표기법(리터럴)
- 단일/이중 따옴표, 연속된 단일/이중 따옴표 세개도 표기법(리터럴)
- 리스트, 튜플, 사전 → 컨테이너(집합체 형태)

❖ 내장 자료형의 정리와 객체 신원 파악

◆ 내장 자료형 알아보기

◉ **type(data)** → data의 자료형을 반환해주는 함수

➡ ex 1]

```
print(type(3))          #정수
print(type(3.3))        #실수
print(type('abc'))      #문자열
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

➡ 설 명 1]

- type(리터럴) → 리터럴의 자료형

➡ ex 2]

```
print(type([]))         #리스트
print(type(()))         #튜플
print(type({}))         #사전(dict)
```

```
<class 'list'>
<class 'tuple'>
<class 'dict'>
```

➡ 설 명 2]

- 내용이 없는 리스트, 튜플, 사전도 타입 조사 가능

❖ 내장 자료형의 정리와 객체 신원 파악

◆ 내장 자료형 알아보기

✣ 자료형 비교

➡ ex 1]

```
a = 0
L = [1,2,3]
print(type(a) == type(0))
print(type(L) == type([]))
print(type(L[0]) == type(0))
```

```
True
True
True
```

➡ 설 명 1]

- type(A)==type(B) → A와 B의 타입이 같은지 확인
- type(L)==type([]) → L과 리스트의 타입이 같은지 확인
- type(L[0]) → L의 0번 인덱스(1)의 타입 → 정수(int)
- type(0) → 0의 타입 → 정수(int)
- type(L[0]) == type(0) → True

❖ 내장 자료형의 정리와 객체 신원 파악

◆ 내장 자료형 알아보기

✣ 자료형 비교

➡ ex 2]

```
print(type(None))
a = None
print(a)
print(type(a))
```

```
<class 'NoneType'>
None
<class 'NoneType'>
```

★None : 아무 값도 없다(혹은 아니다)를 나타내는 객체
★None 타입은 대소문자를 구분해서 사용해야 한다.(소문자로 사용하면 에러)

➡ 설 명 2]

- None 객체는 None 타입
- a = None → a는 아무 것도 아닌 객체
- print(a) → None 출력됨
- print(type(a)) → a의 타입을 출력 → None타입
- None 객체는 None 타입, None 타입은 None 객체

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 내장 자료형의 정리와 객체 신원 파악

◆ 객체의 신원 식별

- **id() : 객체의 식별자를 반환한다.**

➡ ex 1]

```
a = 500
b = a
print(id(a))
print(id(b))
x = 1
y = 1
print(id(x))
print(id(y))
```

```
4478569376
4478569376
4298182776
4298182776
```

➡ 설 명 1]

- id(a) → a 객체의 식별자 반환
- print(id(a)) → 파이썬 인터프리터가 관리하는 식별자 출력
- print(id(b)) → b의 식별자 출력
- b = a 문장으로 인해 b도 500을 가리킴
- a = 500 → 500을 가리키는 레퍼런스가 a에 할당
- b = a → b에도 동일한 레퍼런스가 할당됨
- 따라서 a, b가 동일한 객체(500)를 가리키게 됨
- 때문에 a와 b의 식별자가 동일하게 나타남
- 파이썬에서는 숫자를 따로 객체로 만들지 않음(이미 존재)
- x, y에 1을 가리키는 레퍼런스 할당
- 따라서 x, y는 동일한 객체(1)를 가리킴

❖ 내장 자료형의 정리와 객체 신원 파악

◆ 객체의 신원 식별

- **is : 두 객체의 식별자가 동일한지 테스트**

➡ ex 1]

```
c = [1,2,3]
d = [1,2,3]
print(c is d)
a = 500
b = a
print(a is b)
x = 1
y = 1
print(x is y)
e = f = [4,5,6]
print(e is f)
```

```
False
True
True
True
```

➡ 설 명 1]

- c와 d가 동일한 리스트를 가질 경우
- c is d→ c, d가 가리키는 객체의 식별자가 동일한지 확인
- 결과는 False(식별자가 다름)
- id 함수로 c, d의 식별자를 확인해보면
- 식별자가 다른 c, d → is 함수(id가 동일한가?) → 결과 False 출력
- 내용이 같은 리스트라도 서로 다른 객체
- 수치는 새로 만들어지지 않고 이미 존재
- 이미 존재하는 수치를 c, d가 같이 참조
- 때문에 c, d 식별자가 동일함
- 이미 존재하는 500을 a가 가리킴
- b가 a와 같은 레퍼런스를 가짐 → b도 동일한 500을 가리킴
- f=[4, 5, 6] → [4, 5, 6]을 가리키는 레퍼런스 값이 f에 할당
- e = f → f에 할당된 레퍼런스 값이 e에도 할당
- 따라서 e와 f는 동일한 식별자를 가짐

❖ 내장 자료형의 정리와 객체 신원 파악

◆ 객체의 신원 식별

- == 연산자 : 두 객체의 값이 동일한지를 테스트

➡ ex 2]

```
c = [1,2,3]
d = [1,2,3]
print(c == d)
```

True

➡ 설 명 2]

- c==d → c와 d의 내용이 같은가를 확인
- c와 d는 서로 다른 객체를 가지지만 내용이 같으므로 True
- ==는 두 객체의 내용이 같은지 확인, is는 식별자가 같은지 확인

Python 제어문

❖ 조건문

◆ 정의

- 조건식의 결과를 평가한 후 실행할 내용을 선택해서 실행
- 조건식의 결과는 참 또는 거짓으로 평가 되어야 하고 다음의 경우에는 거짓으로 평가된다.
  - False
  - None
  - 숫자 0 (예] 0, 0.0 등)
  - 비어있는 순서열 (예] '', [], () 등)
  - 비어있는 딕셔너리 (예] {})

◆ 종류

- if 문
- if ~ else 문
- if ~ elif ~ else 문

❖ 조건문

◆ if 문

- 조건식의 결과가 참이면 내용을 실행하는 제어문

✣ 형식

```
if 조건식:
  실행문
```

➡ ex 1]

```
no = int(input('정수를 입력하세요!'))
if (no % 2 == 0):
    print('입력한 수 [ {0} ] 는 짝수 입니다!'.format(no))

if (no % 2 != 0):
    print('입력한 수 [ {0} ] 는 홀수 입니다!'.format(no))
```

```
정수를 입력하세요!11
입력한 수 [ 11 ] 는 홀수입니다.
```

➡ 설 명 1]

- 변수 no에 입력한 숫자를 Number 타입으로 변환후 저장하고 if 문에서 조건식을 평가한다.
- 첫번째 if문에서는 no를 2로 나눈 나머지가 0과 같으면 실행
- 두번째 if문에서는 no를 2로 나눈 나머지가 0과 같지 않을 경우 실행

✤ 참고

◆ 출력 형식을 정해서 출력하기

✤ 형식

```
print('내용 {인덱스1}  내용 {인덱스2} ... 내용'.format(data1, data2, ... , dataN))
```

➡ 설 명 1]

- 데이터의 타입 구분없이 format() 에 입력해주는 매개변수의 순서대로 인덱스를 사용하면 된다.

❖ 조건문

◆ if ~ else 문

- 조건식의 결과가 참일 경우와 거짓일 경우의 실행내용을 정의해서 실행하는 제어문

✣ 형식

```
if 조건식:
   실행문
else:
   실행문
```

➡ ex 1]

```
no = int(input('정수를 입력하세요!'))
if (no % 2 == 0):
    print('입력한 수 [ {0} ] 는 짝수 입니다!'.format(no))
else:
    print('입력한 수 [ {0} ] 는 홀수 입니다!'.format(no))
```

```
정수를 입력하세요!11
입력한 수 [ 11 ] 는 홀수입니다.
```

➡ 설 명 1]

- 조건식이 참일 경우는 if 절의 내용을 실행하고 거짓일 경우는 else절의 내용을 실행한다.

❖ 조건문

◆ if ~ elif ~ else 문

- 두가지 이상의 조건식의 결과가 참일 경우의 실행내용을 정의해서 실행하는 제어문

✤ 형식

```
if 조건식1:
   실행문1
elif 조건식2:
   실행문2
   ...
elif 조건식n:
   실행문n
else:
   실행문
```

❖ 조건문

◆ if ~ elif ~ else 문

➡ ex 1]

```
no = int(input('정수를 입력하세요!'))
if (no == 0):
    print('입력한 수는 0 입니다!')
elif (no % 2 == 0)
    print('입력한 수 [ {0} ] 는 짝수 입니다!'.format(no))
else:
    print('입력한 수 [ {0} ] 는 홀수 입니다!'.format(no))
```

```
정수를 입력하세요!11
입력한 수 [ 11 ] 는 홀수입니다.
```

❖ 반복문

◆ 정의

- 프로그램의 흐름을 반복해서 되풀이하는 문장(명령)
- 루프문(Loop Statement) 이라고도 한다.

◆ 종류

- while 문
- for 문

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 반복문

◆ range()

입력받은 숫자에 해당되는 범위의 값을 반복 가능한 객체로 만들어 반환

✤ 형식

range([start,] stop[, step])

➡ ex 1]

```
# range(stop) : 0 ~ stop-1 까지의 정수
print(range(5), type(range(5)))
print(tuple(range(5)))
print(set(range(5)))
print(list(range(5)))

# range(start,stop) : start ~ stop-1 까지의 정수
print(list(range(5, 10)))

# range(start,stop, step) :
#                   start 부터 step 씩 증감하며 stop-1 까지의 정수
print(list(range(2, 11, 2)))
print(list(range(1, 11, 2)))
print(list(range(9, 0, -2)))
print(list(range(10, 0, -2)))
```

```
range(0, 5) <class 'range'>
(0, 1, 2, 3, 4)
{0, 1, 2, 3, 4}
[0, 1, 2, 3, 4]
[5, 6, 7, 8, 9]
[2, 4, 6, 8, 10]
[1, 3, 5, 7, 9]
[9, 7, 5, 3, 1]
[10, 8, 6, 4, 2]
```

훈련교사 : 전 은 석
email : euns_jun@naver.com

❖ 반복문

◆ enumerate()

- enumerate는 “열거하다”라는 뜻
- 리스트가 있는 경우 순서와 리스트의 값을 전달하는 기능
- 이 함수는 순서가 있는 자료형(list, set, tuple, dictionary, string)을 입력으로 받아 인덱스 값을 포함하는 enumerate 객체를 리턴
- 보통 enumerate 함수는 for문과 함께 자주 사용

➡ ex 1]

```
data = enumerate((1, 2, 3))
print(data, type(data))

for i, value in data:
    print(i, ":", value)

data = enumerate({1, 2, 3})
for i, value in data:
    print(i, ":", value)

data = enumerate([1, 2, 3])
for i, value in data:
    print(i, ":", value)

dict1 = {'이름': '한사람', '나이': 33}
data = enumerate(dict1)
for i, key in data:
    print(i, ":", key, dict1[key])

data = enumerate('재미있는 파이썬')
for i, value in data:
    print(i, ":", value)
```

```
<enumerate object at 0x0000000002424EA0> <class 'enumerate'>
0 : 1
1 : 2
2 : 3
0 : 1
1 : 2
2 : 3
0 : 1
1 : 2
2 : 3
0 : 이름
1 : 나이
0 : 재
1 : 미
2 : 있
3 : 는
4 :
5 : 파
6 : 이
7 : 썬
```

❖ 반복문

◆ while 반복문

- 조건이 참인 동안 반복 실행

✤ 형식

```
while 조건:
    조건이 참인동안 실행할 명령들....
```

・break : 반복문을 탈출
・continue : 반복문의 처음으로 보낸다.

- 참인 경우에 실행할 명령들은 들여쓰기를 한다. 대부분 4칸 들여쓰기를 합니다.
- 들여쓰기가 끝나면 반복문을 종료
- 조건이 거짓이면 아무런 일도 하지 않는다.
- 반복문은 얼나든지 중첩이 가능합니다. 중첩할때 들여쓰기에 주의
- 반복문 내에서 break 명령을 만나면 현재의 반복문을 탈출
- 반복문 내에서 continue 명령을 만나면 현재의 반복문의 처음으로 돌아가서 조건을 비교

❖ 반복문

◆ while 반복문

➡ ex 1]

```
i = 1
dan = 2
while i < 10:
    print('{0} x {1} = {2}'.format(dan, i, dan*i))
    i += 1
```

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
```

❖ 반복문

✣ 문자열 형식화

✣ 형식

```
"출력형식".format(데이터, 데이터, ...)
```

◉ 길이와 정렬

- {:길이} : 출력할 데이터의 길이를 지정합니다. 문자열(왼쪽 정렬), 숫자(오른쪽 정렬)
- {:<길이} : 왼쪽 정렬
- {:>길이} : 오른쪽 정렬
- {:^길이} : 가운데 정렬

◉ 채움문자와 숫자 표시형식

- {[인덱스]:[채움문자][정렬][길이][,|_][형식문자]
- 공백을 채움문자로 채워줍니다.
- , : 천단위 마다 콤마를 붙여줍니다.
- _ : 천단위 마다 밑줄을 붙여줍니다.
- e : 숫자를 지수 형식으로 만들어 줍니다.
- = : 숫자 형식에만 사용합니다. 부호를 항상 제일 앞에 출력합니다.

❖ 반복문

✣ 문자열 형식화

✣ 형식

◉ 진법과 실수 출력 형식지정

- b(2진수), o(8진수), x(16진수 소문자), X(16진수 대문자)
- {[인덱스]:[전체자릿수][.소수이하자릿수]f

◉ {}와 % 출력

- {}를 출력하려면 "{"과 "}"을 세번 연달아 입력해야 합니다.
- % 자체를 출력하려면 %%로 입력해야 합니다.

➡ ex 1]

```
print('{{{0}}}씨는 상위 {{{1}}}%안에 있는 사람입니다.'.format('한사람',10))
print('%s씨는 상위 %d%%안에 있는 사람입니다.'%('한사람',10))
```

```
한사람씨는 상위 10%안에 있는 사람입니다.
한사람씨는 상위 10%안에 있는 사람입니다.
```

✣ 참고 - 문자열 앞에 0으로 채우기

```
문자열.zfill(숫자)
```

- 문자열을 지정한 숫자만큼 공간을 확보하고 남는 앞부분을 "0"으로 채움

❖ 반복문

✤ 문자열 형식화

✤ 형식 정리

{인덱스:[[fill] align] [sign] [#] [0] [width] [grouping_option] [. precision] [type]}

★ [ ] 는 생략 가능

- fill(채움문자) : 어떠한 문자도 가능합니다.
- align(정렬) : <(왼쪽정렬, 문자열 기본값), >(오른쪽 정렬, 숫자 기본값), ^(가운데정렬)
- =(숫자에만 사용, '+000000120'형식으로 필드를 인쇄하는 데 사용)
- sign(부호) : +(양수도 부호 표시), -(음수에 대해서만 부호를 사용,기본값)
- width(폭) : 전체 폭을 양의 정수로 지정합니다.
- , | _ : 천단위마다 콤마(,) 또는 밑줄(_)을 붙여줍니다.
- precision(소수이하 자리수) : 소수이하 자리수를 정수로 지정합니다.
- type(형식) : 문자열(s), 정수(b, c, d, o, x, X, n), 실수(e, E, f, F, g, G, n)

❖ 반복문

◆ for 반복문

✣ 형식

```
for 변수명 in Collection_Data:
        반복 실행할 명령
        ....
[else:
        데이터가 없을때 실행할 명령
        ....]
```

- in 뒤의 Collection_Data 값이 없을때까지 하나씩 변수로 복사되어 반복
  - Collection_Data : List, Tuple, Set, Dictionary, range(), enumerate()
- Collection_Data로는 tuple, set, list, dictionary, string 등 데이터의 집합
- else 절은 선택 사항

❖ 반복문

◆ for 반복문

➡ ex 1]

```
no = [1, 2, 3]
for n in no:
    print(n)
else:
    print('실행종료')
```

```
1
2
3
실행종료
```

➡ 설 명 1]

- 리스트 no 에 들어있는 데이터를 하나씩 꺼내서 출력
- 모두 꺼내서 출력한 후 else 절 실행

❖ 반복문

◆ for 반복문

➜ ex 2]

```
for i in (1, 2, 3):
    print(i, end=' ')
print()

for i in {1, 2, 3}:
    print(i, end=' ')
print()

for i in [1, 2, 3]:
    print(i, end=' ')
print()

dict1 = {'이름': '한사람', '나이': 33}
for i in dict1:
    print(i, ':', dict1[i])
print()

for i in 'Python':
    print(i)
print()

for i in []:
    print(i)
else:
    print('데이터가 없습니다. ')
```

```
1 2 3

1 2 3

1 2 3

이름 : 한사람
나이 : 33

P
y
t
h
o
n

데이터가 없습니다.
```

❖ 반복문

◆ for 반복문

✣ for 반복문을 이용한 리스트 만들기

[표현식 for 변수 in Collection_Data]
[표현식 for 변수 in Collection_Data if 조건]

➡ ex 1]

```
list1 = [i**2 for i in range(1,11)]
print(list1)

list2 = [i for i in range(1,11) if i%2]
print(list2)

list3 = [i for i in range(1,11) if not i%2]
print(list3)

# 중첩이 가능
#구구단표
list4 = [i*j for i in range(1,10)
           for j in range(1,10)]

for i in range(0,len(list4)):
    print('{:3d}'.format(list4[i]), end=' ')
    if (i+1)%9==0:
        print()
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[1, 3, 5, 7, 9]
[2, 4, 6, 8, 10]
  1  2  3  4  5  6  7  8  9
  2  4  6  8 10 12 14 16 18
  3  6  9 12 15 18 21 24 27
  4  8 12 16 20 24 28 32 36
  5 10 15 20 25 30 35 40 45
  6 12 18 24 30 36 42 48 54
  7 14 21 28 35 42 49 56 63
  8 16 24 32 40 48 56 64 72
  9 18 27 36 45 54 63 72 81
```