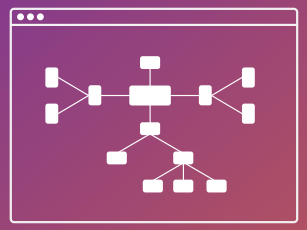
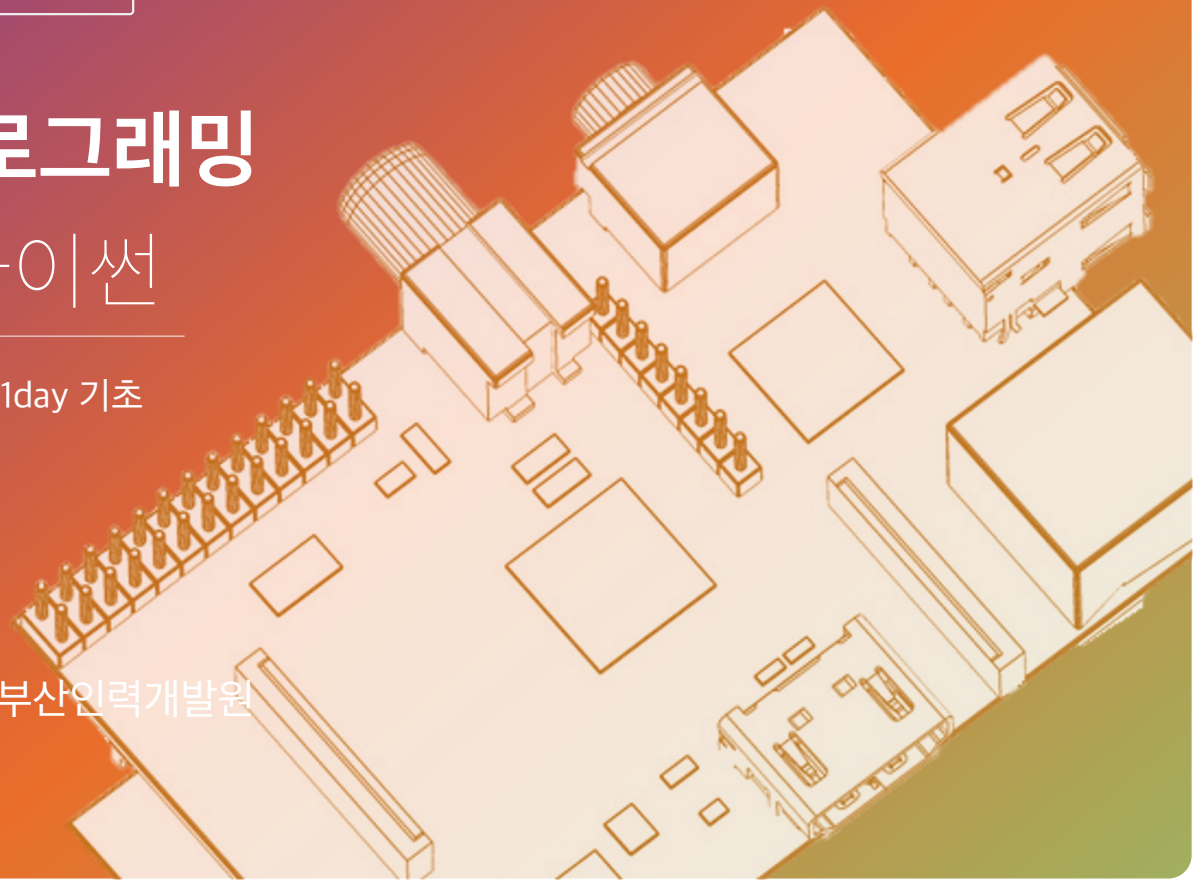


로봇 프로그래밍

AI+파이썬

03_파이썬 1day 기초



대한상공회의소 부산인력개발원

강 의 일 정

IoT 기반 공정설비모니터링 프로그래밍(5d)　　로봇 프로그래밍(AI+파이썬)(3d)

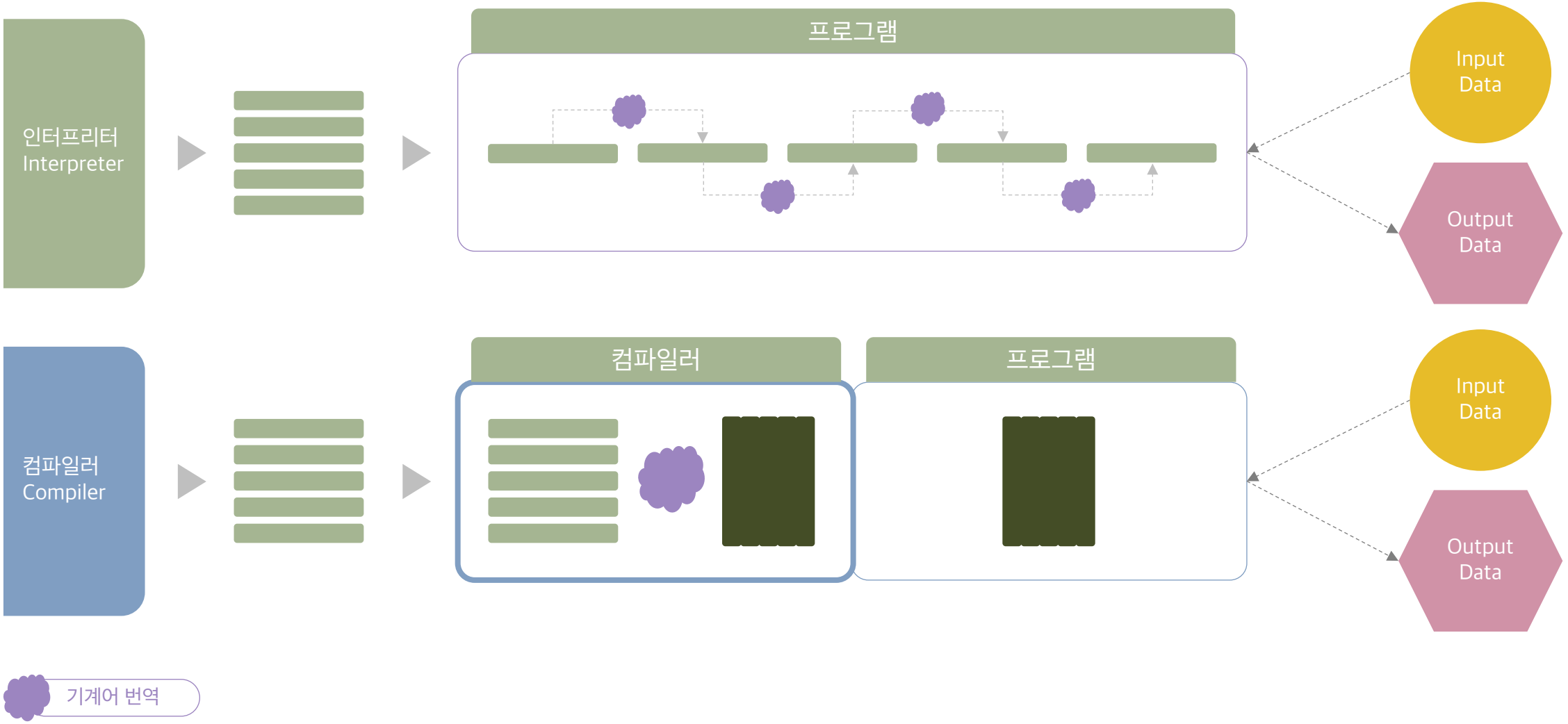
1회 차 8월 19일(월)	IoT 기반 공정설비 모니터링의 이해	IoT의 이해 / 센서 & 모니터링 시스템 / IoT네트워크	이론
2회 차 8월 20일(화)	라즈베리파이 & 센서 프로그래밍 1	Rpi의 이해 / Rpi 개발환경 구축 / 센서 프로그래밍(LED/온·습도/초음파…)	이론 /실습
3회 차 8월 21일(수)	라즈베리파이 & 센서 프로그래밍 2	Rpi 원격 개발환경 / 센서 응용 프로그래밍	이론 /실습
4회 차 8월 22일(목)	Node-Red 제어&모니터링 프로그래밍 1	Node-Red의 이해 / 개발환경 구축 / 센서 모니터링 및 제어	이론 /실습
5회 차 8월 26일(월)	Node-Red 제어&모니터링 프로그래밍 2	Dashboard 설계 / http통신 & DB / MQTT	이론 /실습
6회 차 8월 27일(화)	ESP32 마이크로 파이썬 로봇 프로그래밍 1	ESP32의 이해 / 마이크로 파이썬 개발환경 구축 / 로봇 프로그래밍(LEC/온·습도/초음파…)	이론 /실습
7회 차 8월 28일(수)	ESP32 마이크로 파이썬 로봇 프로그래밍 2	로봇 프로그래밍(DC모터/서보모터/LCD…) / 웹 서버 구축 및 제어 / AI 프로그래밍 이해	이론 /실습
8회 차 9월 4일(수)	파이썬 프로그래밍	1day 파이썬 기본문법	이론 /실습

CONTENTS

업무를 시작하기 위해

좋은 목수는 바로 집을 짓지 않는다.
장비를 점검하고, 보강한 다음
머리속에서 충분한 시간을 들여 진행과정을 고민하고,
발생할 문제점을 충분히 검토한 이후 에서야
작업을 시작한다.

컴파일러 vs 인터프리터



파이썬 소개

1991년에 귀도 반 로섬(Guido van Rossum)이 개발한 대화형 프로그래밍 언어

대화형 구조로
생산성이 높음

간결하면서도
효율적인 프로그래밍
빠르게 작성

파이썬 변수(Variable)

변수(Variable)

변수 또는 스칼라는 컴퓨터 프로그래밍 과정에서 발생하는 데이터를 의미하며,
메모리에 담겨 연산에 사용되거나 연산이 완료된 결과값으로 반환되는 데이터를 뜻한다.

메모리 주소	데이터
00120A08	01011110010
00120A09	10011001010
00120A10	10111000011
00120A11	10101101101
00120A12	01111100110

변수 명	데이터
x	15
y	16
z	17
label	sum
sum	16

객체

선언

할당

변수 명(Variable Name) 표기

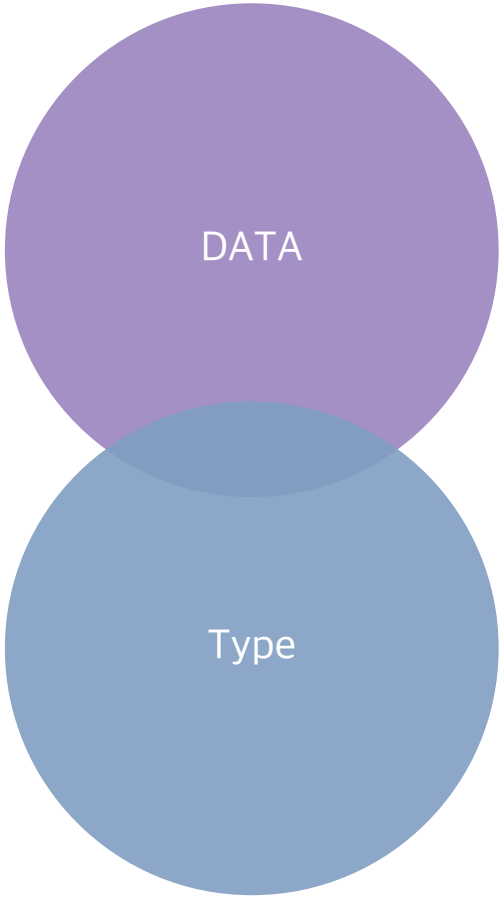
카멜 표기법(Camel Case)

카멜 표기법 또는 낙타 표기법은 프로그래밍에서 파일, 변수, 함수 등 대상의 이름을 띄어쓰기 없이 표기하기 위해 따르는 네이밍 컨벤션
단어 전체적으로 소문자를 사용하지만 맨 첫 글자를 제외한 각 합성어의 첫 글자만 대문자 표기하는 방법.
(올 캡스 / 스네이크 표기법)

변수 명 작성 규칙

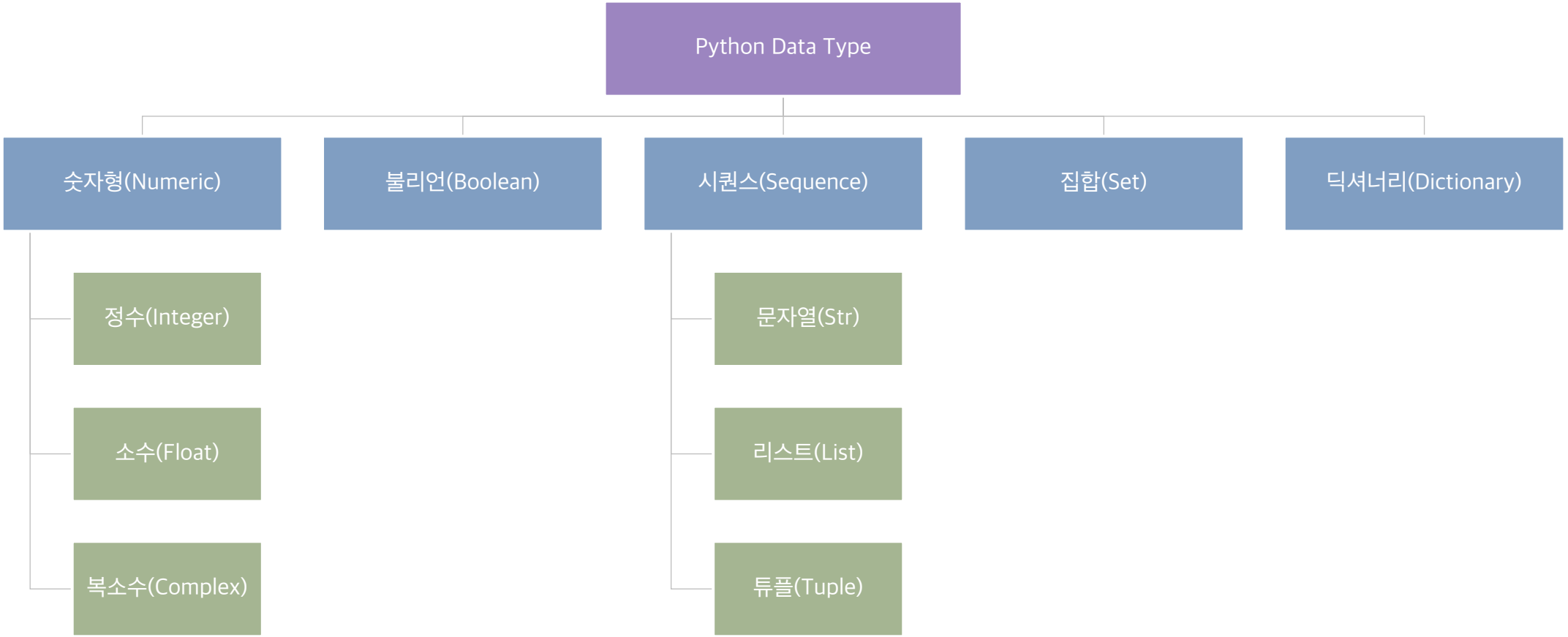
- 변수명에는 특수문자 사용 불가
- 언더 바(_)는 특수문자지만 변수명에 사용 가능
- 영문과 숫자를 혼합해서 작성 가능
- 한글 변수 명 가능
- 변수명에 숫자 사용시 첫 글자는 사용 불가
- 변수명은 대소문자를 구별(variable, Variable은 다른 변수)
- 프로그래밍 언어에 따라 금지된(사전 정의 된)변수는 사용 불가

DATA vs DATA Type



• 문자 (String)	>name = “python”
• 숫자 (Numeric)	>x = 1991
• 부울리언 (Boolean)	>cording_easy = True
• 리스트 (List)	>programing = [“c”,”python”,”react”]
• 튜플 (Tuple)	>programing = (“c”,”python”,”react”)
• 딕셔너리 (Dictionary)	>programing = {‘compiler’:’c’,’interpreter’:’python’}
• 집합 (Set)	>programing = {‘c’, ‘python’, ‘react’}

Python Data Type



문자열(Str)

- 문자열은 단어, 문장 과 같은 일련의 텍스트 데이터를 나타내는데 사용.
- 문자열은 작음 따옴표('), 큰 따옴표("")감싸져있으며, 따옴표 안의 문자들이 문자열로 정의.
- 문자열은 시퀀스 유형에 포함되어 있는 이유는, 문자들이 일련의 순서로 배열되어 있기 때문

```
#문자열
name = “python”
message = “Hello, Python”
```

숫자형(Numeric)

숫자형

- 숫자형은 정수(Integer), 실수(Float), 복소수(Complex) 로 구분.
- 정수는 양수, 음수, 혹은 0이 될 수 있는 숫자.
- 실수는 소수점이 있는 숫자이며, 복소수는 실수와 허수 요소를 가진 숫자.
- **숫자형은 산술 연산을 비롯 다양한 연산 수행에 사용.**

```
#정수
x = 42
y = -10

#실수
x = 3.14
y = 1.0

#복소수
z = 3 + 4j
```

부울리언(Boolean)

- 부울리언(Boolean)은 True와 False의 두 가지 값을 갖는 자료형.
- 부울리언 값은 진실 혹은 거짓을 나타내기 때문에 프로그래밍에서 조건을 판단할 때 많이 사용.
- if문이나 while문에서 조건식으로 부울리언 값을 사용하여 프로그램 흐름을 제어.
- 두 값의 비교 결과로도 부울리언 값을 얻음.

```
#부울리언
python_hard = False
python_easy = True

#값 비교(결과값에 True할당)
x = 5
y = 10
result = x < y
```

리스트(List)

리스트

- 리스트는 다수의 데이터를 저장하는데 사용.
- 리스트에 포함된 데이터는 "요소(Element)".
- 리스트에서 각 요소는 고유한 위치(Index)를 가지고 있고, 이 위치를 통해 개별 요소에 접근.
- 리스트는 대괄호([])를 사용하여 정의하며, 동일한 자료형의 요소를 가질 수도 있지만, 다른 자료형의 요소를 가질 수도 있음.


```
#리스트
programings = [“c”, “python”, “react”]

first_programing = programings[0] #“c”
second_programing = programings[1] #“python”
third_programing = programings[2] #“react”
```

튜플(Tuple)

튜플

- 튜플은 리스트와 매우 유사하지만, 한 번 생성된 튜플의 요소는 변경할 수 없음.
- 튜플은 불변의 데이터 구조를 지녀 안정성이 중요한 상황에서 사용.
- 튜플은 괄호 (())를 사용하여 정의.


```
#리스트
programing = [“c”, “python”, “react”]

first_programing = programings[0] #“c”
second_programing = programings[1] #“python”
third_programing = programings[2] #“react”
```

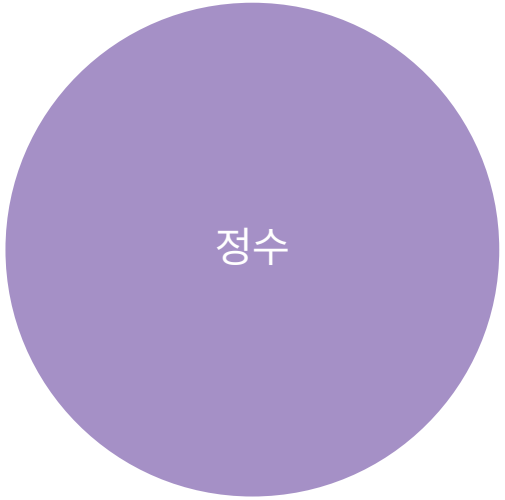
집합(Set)

- 집합은 중복된 값이 없는 유일한 값들의 모임.
- 집합은 순서가 없는 데이터 구조이므로, 인덱스로 접근할 수 없으나, 집합에 특정 값이 포함되어 있는지 확인가능.
- 집합은 다양한 연산을 지원하는 기능을 포함. 예를 들어, 교집합, 합집합, 차집합 등을 연산가능.
- 집합은 중괄호 ({})를 사용하여 정의할 수 있습니다.

```
#딕셔너리
programings = {‘c’, ‘python’, ‘react’, ‘c’} #{‘c’, ‘python’, ‘react’}

Print(‘python’ in programings) #True
Print(‘c#’ in programings) #False
```

지금은?

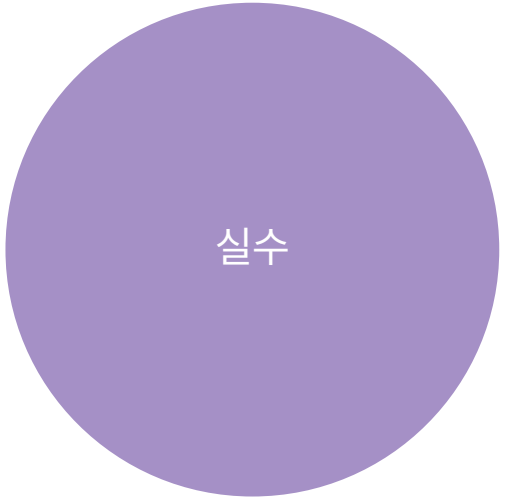
정수(Integer)



- 정수는 Integer의 약자인 int를 사용하여 정수형을 나타내는 자료형.
- 정수는 아래와 같이 -1000, 250, -10 ...과 같이 양의 정수, 음의 정수를 표현.

```
#정수
>>> money = -1000
>>> volume = 250
>>> count = -10
>>> day = 2
>>> month = 9
>>> year = 12

>>> a = -5
>>> b = 7
>>> c = 10
```

실수(Float)



- 실수(real number)는 floating point의 약자인 float를 사용하여 실수형을 나타내는 자료형.
- 정수 데이터에는 소수점이 없고 실수 데이터에는 소수점이 존재.
- 실수는 아래와 같이 177.5, -1.23 처럼 소수점을 표현.
- 숫자 계산을 하는 경우 계산에 사용된 숫자 중 하나라도 소수점이 있으면 계산결과는 실수로 처리.
- 나눗셈 연산의 결과는 입력에 상관없이 항상 실수로 처리.

```
#실수
>>> tall = 177.5
>>> pi = 3.1415926
>>> a = -1.23
>>> b = 1.23
```

부동소수점(Floating Point)

부동소수점

- 부동 소수점 방식은 대부분 IEEE 754라는 국제표준에 따라 표현.
- 부동소수점 방식에서는 숫자를 정수로 된 유효숫자와 정수로 된 지수의 곱으로 표현.
- 예를 들어 십진수 부동소수점 방식에서 123.456이란 숫자는 123456×10−3123456×10^{-3}이므로 123456이라는 정수 유효숫자와 $^{-3}$이라는 정수 지수로 나타낼 수 있음.
- 파이썬에서는 유효숫자e지수 라는 방법으로 부동 소수점 형태를 직접 표현합니다.

```
#부동소수점
# 123e2 = 123.0 * 100 = 12300.0
>>> 123e2

# 123e-2 = 123.0 * 0.01 = 1.23
>>> 123e-2

# 123.456e-3 = 123.456 * 0.001 = 0.123456
>>> 123.456e-3
```

사칙연산

사칙연산

```
#사칙연산 변수 설정
>>> a = 3
>>> b = 5

>>> a + b#더하기 연산
8

>>> a - b #빼기 연산
-3

>>> a * b #곱하기 연산
15

>>> a / b #나누기 연산
0.6
```

기타연산

기타연산

```
#기타연산 변수설정
>>> a = 5
>>> b = 2

>>> a ** b #제곱연산
25

>>> a % b #나머지 반환 연산
1

>>> a // b #몫을 반환하는 연산
2
```

문자열(Str)

문자열

- 문자열을 따옴표를 사용하여 표기.
- 문자열을 표기할 때는 시작 따옴표와 마침 따옴표가 반드시 동일.
- 숫자의 경우 따옴표로 표기하면 문자로 인지.

```
#큰따옴표 표기
>>> "Hello, Python"
>>> "2024-09-01"

#작은따옴표 표기
>>> 'Hello, Python'
>>> '2024-09-01'

#큰따옴표 3개(""")표기
>>> """Hello, Python"""
>>> """2024-09-01"""

#작은따옴표 3개(''')표기
>>> '''Hello, Python'''
>>> '''2024-09-01'''
```

문자열(Str) 줄바꿈

이스케이프 코드
줄바꿈

```
#이스케이프 코드 \n 사용
>>> "안녕하세요. \n 반갑습니다. \n 저는 Python을 공부합니다."

안녕하세요.
반갑습니다.
저는 Python을 공부합니다.
```

문자열(Str) 줄 바꿈

3중 따옴표
줄 바꿈

```
#3중 따옴표(""") 줄 바꿈
>>> """
        안녕하세요.
        반갑습니다.
        저는 Python을 공부합니다.
        """

안녕하세요.
반갑습니다.
저는 Python을 공부합니다.
```

문자열(Str) 포매팅

% 포매팅

```
#Str
>>> tmpSting = "맑음"
>>> "오늘 날씨는 %s 입니다."
오늘 날씨는 맑음 입니다.

#Int
>>> tmpInt = 27
>>> "오늘 온도는 %d 도입니다."
오늘 온도는 27 도 입니다.

#Persen
>>> tmpPersen = 54
>>> "오늘 습도는 %d%% 입니다." #'%' 표기를 할 경우 %%를 사용
오늘 습도는 54% 입니다.

#Float
>>> tmpFloat = 20.5
>>> "오늘 강수확률은 %f%% 입니다."
오늘 강수확률은 20.5% 입니다.
```

문자열(Str) 포매팅

Format활용
포매팅

```
>>> tmpString = "흐림"
>>> "오늘 날씨는 {} 입니다.".format(tmpString)
오늘 날씨는 흐림입니다.

>>> tmpInt = 177
>>> "저의 키는 {} 입니다.".format(tmpInt)
저의 키는 177입니다.

>>> tmpFloat = 0.65
>>> "오차 값은 {} 입니다.".format(tmpFloat)
오차 값은 0.65입니다.

>>> tmpArrow = "값"
>>> "괄호안에 {{{}}}을 넣어주세요.".format(tmpArrow)
# 문자 {를 사용하고 싶은 경우 {{를 이용해 사용할 수 있습니다.
# 문자 }를 사용하고 싶은 경우 }}를 이용해 사용할 수 있습니다.
괄호안에 {값}을 넣어주세요.
```

문자열(Str) 포매팅

```
>>> "이름은 {0}입니다. 점수는 {1}점 입니다. 그러므로 학점은 {2}입니다.".format("홍길동",95,"A+")
이름은 홍길동입니다. 점수는 95입니다. 그러므로 학점은 A+입니다.

>>> "이름은 {name}입니다. 점수는 {score}점 입니다. 그러므로 학점은 {Credit}입니다.".format(name ="홍길동", score =95, Credit = "A+")
이름은 홍길동입니다. 점수는 95점 입니다. 그러므로 학점은 A+입니다.

>>> "이름은 {0}입니다. 점수는 {score}점 입니다. 그러므로 학점은 {Credit}입니다.".format("홍길동", score =95, Credit = "A+")
이름은 홍길동입니다. 점수는 95점 입니다. 그러므로 학점은 A+입니다.
```

문자열(Str) 포매팅

f-string활용
포매팅

```
>>>name = "홍길동"
>>>score = 95
>>>credit = "A+"
>>>print(f"이름은 {name}. 점수는 {score}점 입니다. 그러므로 학점은 {credit}입니다.")

이름은 홍길동. 점수는 95점 입니다. 그러므로 학점은 A+입니다.
```

지금은?

리스트(List)

리스트

- 리스트(List)는 파이썬의 기본적인 자료형 중 하나로, 여러 개의 값을 담을 수 있는 순차적인 자료 구조.
- 리스트에는 숫자, 문자열, 부울리언 값, 또는 다른 리스트 등을 담을 수 있음.
- 여러 개의 **요소(element)**를 가지고 있고, 각 요소는 순서가 있으며, **인덱스(Index)**값을 가짐.
- 리스트의 첫 번째 요소의 인덱스는 0에서 시작.

리스트의 **장점**

- 다양한 자료형을 포함.
- 순차적인 자료 구조이므로, 쉽게 요소를 추가, 삭제, 수정.
- 인덱스를 이용하여 요소에 접근할 수 있어, 빠른 접근이 가능.
- 내장 함수를 이용하여 리스트의 정렬, 검색, 길이 계산 등을 쉽게 수행.

리스트의 **단점**

- 메모리 사용량이 많아질 수 있다.
- 요소에 접근할 때, 모든 요소를 검색하여 인덱스를 찾아야 하므로, 탐색 속도가 느릴 수 있다.
- 삭제, 추가, 수정 작업에서 리스트의 길이가 길어지면, 비효율적일 수 있다.

리스트(List)

리스트 생성

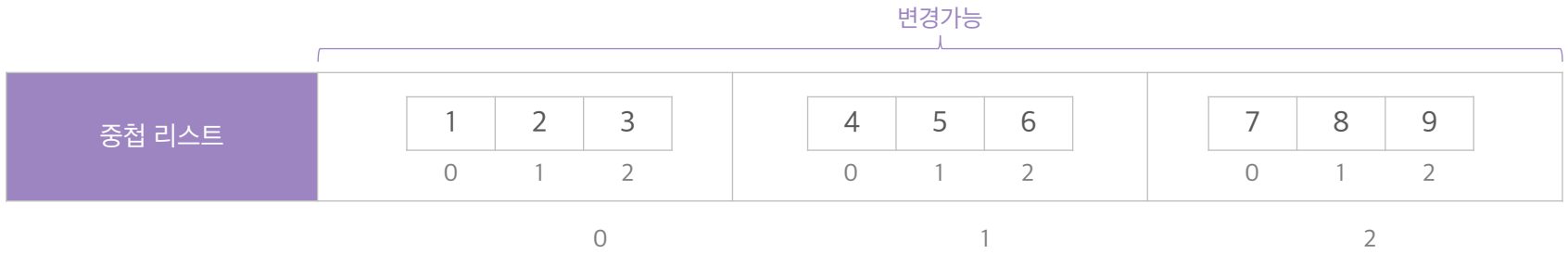
- 리스트는 대괄호([ ])를 이용하여 생성.
- 리스트의 각 요소는 쉼표(,)로 구분

```
#기본적인 생성방법
>>>numbers = [1, 2, 3, 4, 5]

#다형 자료 생성방법
>>>mixed_list = [1, “Python”, 3.14, True]

#빈 리스트
>>>empty_list = []

#list()이용 빈 리스트
>>>empty_list = list()
```

중첩 리스트(Nested List)

중첩 리스트 생성

- 중첩 리스트(Nested List)를 활용하여 리스트 내 다른 리스트를 포함 가능.




```
#중첩 리스트(행)
>>>nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

#중첩 리스트(열)
>>>mixed_list = [
                    [1, 2, 3]
                    [4, 5, 6]
                    [7, 8, 9]
                ]
```

리스트 인덱스(List Index)

리스트 인덱스

- 파이썬에서는 리스트의 각 요소에 인덱스(Index)를 할당하고 특정 위치에 있는 요소를 추출.
- 인덱스는 0부터 시작.

변경가능

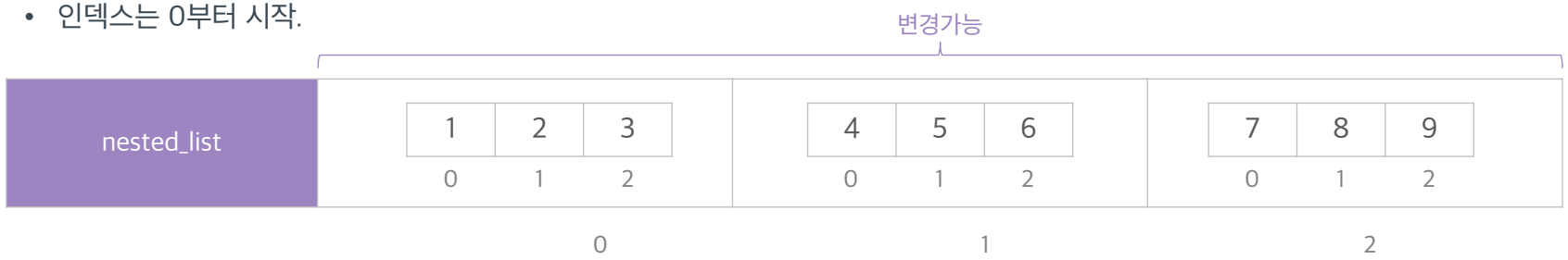
my_list	1 (Index : 0)	2 (Index : 1)	3 (Index : 2)	4 (Index : 3)	5 (Index : 4)

```
#리스트 인덱스로 값 추출
>>> my_list = [1, 2, 3, 4, 5]

>>> print(my_list[0])
1
>>> print(my_list[2])
3
```

중첩 리스트 인덱스(Nested List Index)

중첩 리스트
인덱스

- 파이썬에서는 리스트의 각 요소에 인덱스(Index)를 할당하고 특정 위치에 있는 요소를 추출.
- 인덱스는 0부터 시작.

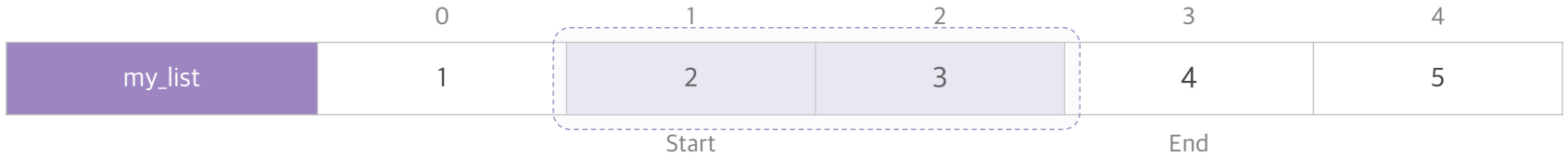



```
#중첩 리스트 인덱스로 값 추출
>>> nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>> print(nested_list[0][0])
1

>>> print(nested_list[2][1])
8
```

리스트 슬라이싱(List Slicing)

슬라이싱

- 리스트에서 여러 개의 요소를 한꺼번에 추출하려면, 슬라이싱(slicing)을 사용.
- 슬라이싱은 문자열에서 일부분을 추출하는 것을 의미.
- Start는 추출하고자 하는 요소의 시작 위치를, End는 끝 위치를 지정하면 Start와 End 사이의 요소들이 추출.
- Step은 생략이 가능하며, 생략되면 기본 값 1로 적용.




```
#리스트 슬라이싱
>>> list[start:end:step]

>>> my_list = [1, 2, 3, 4, 5]
>>> print(my_list[1:3])
[2, 3]
```

리스트 슬라이싱(List Slicing)

슬라이싱

- 첫 번째 인덱스를 생략하면, 0부터 시작. 만약, 두 번째 인덱스를 생략하면, 리스트의 마지막 요소까지 추출.
- 첫 번째 슬라이싱인 my_list[:3]에서 :3은 0부터 2까지의 요소를 추출하는 것을 의미, 따라서, [1, 2, 3]이 출력.
- 두 번째 슬라이싱인 my_list[2:]에서 2:은 2부터 마지막 요소까지의 요소를 추출하는 것을 의미. 따라서, [3, 4, 5]이 출력.

	0	1	2	3	4
my_list	1	2	3	4	5

```
#리스트 슬라이싱 Start 생략
>>> print(my_list[:3])
[1, 2, 3]

#리스트 슬라이싱 End 생략
>>> print(my_list[2:])
[3, 4, 5]

#리스트 슬라이싱 간격 지정
>>> print(my_list[0:4:2])
[1, 3]
```

리스트 연산

리스트
더하기

list1	1	2	3

+

list2	4	5	6

=

list3	1	2	3	4	5	6

```
#리스트 더하기
>>> list1 = [1, 2, 3]
>>> list2 = [4, 5, 6]
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4, 5, 6]
```

리스트 연산

리스트
곱하기

list1	1	2

✖ 3 =

list2	1	2	1	2	1	2

```
#리스트 곱하기
>>> list1 = [1, 2]
>>> list2 = list * 3
>>> print(list2)
[1, 2, 1, 2, 1, 2]
```

리스트 연산

리스트
수정 & 삭제

- 리스트의 특정 위치의 요소를 **수정**하려면 해당 위치에 새로운 값 할당.

list1	1	2	3 -> 7	4	5	6

- 리스트의 특정 위치의 요소를 **삭제**하려면 del 명령어로 삭제.

list2	a	b	C -> 삭제	d	e	f

```
#리스트 수정 & 삭제
>>> list1 = [1, 2, 3, 4, 5, 6]
>>> list2 = [a, b, c, d, e, f]
>>> list1[2] = 7 #변수 재할당
>>> del list2[2] #변수 삭제

>>> print(list1)
[1, 2, 7, 4, 5, 6]

>>> print(list2)
[a, b, d, e, f]
```

리스트 함수

len()
함수

- 리스트에 포함 된 요소의 개수 함수

```
#리스트 수정 & 삭제
>>> list = [1, 2, 3, 4, 5, 6]
>>> print(len(list))
6
```

max()
함수

- 리스트 내 가장 큰 값을 찾아주는 함수

```
#리스트 수정 & 삭제
>>> list = [1, 2, 3, 4, 5, 6]
>>> print(max(list))
6
```

리스트 함수

min()
함수

- 리스트 내 가장 작은 값을 찾아주는 함수

```
#리스트 수정 & 삭제
>>> list = [1, 2, 3, 4, 5, 6]
>>> print(min(list))
1
```

sum()
함수

- 리스트 내 값을 모두 더한 결과를 반환하는 함수

```
#리스트 수정 & 삭제
>>> list = [1, 2, 3, 4, 5, 6]
>>> print(sum(list))
21
```

리스트 함수

sorted()
함수

- 리스트에 저장된 값을 정렬하는 함수.
- 오름차순 또는 내림차순 정렬 가능, 정렬순서를 내림차순으로 바꾸기 위해서는 reverse인자를 True로 설정(기본값은 False)
- 정렬된 리스트는 새로운 리스트 객체로 반환되며, 원본 리스트에 영향을 미치지 않음.

list	3	2	5	4	1

sorted(list)

sorted_list	1	2	3	4	5

```
#sorted함수를 활용한 정렬
>>>list = [3, 2, 5, 4, 1]
>>>sorted_list = sorted(list)
>>>print(sorted_list)
[1, 2, 3, 4, 5]

>>>list = [3, 2, 5, 4, 1]
>>>sorted_list = sorted(list, reverse=True)
>>>print(sorted_list)
[5, 4, 3, 2, 1]
```

리스트 메서드

append()
메서드

- 리스트 끝에 새로운 요소를 추가하는 기능

```
#.append()
>>> list = [1, 2, 3]
>>> list.append(4)
>>> print(list)
[1, 2, 3, 4]
```

insert()
메서드

- 리스트 내 새로운 요소를 삽입하는 기능(첫번째 인자는 위치인 인덱스 값, 두번째 인자는 삽입할 요소의 값)

```
#.insert()
>>> list = [1, 2, 3]
>>> list.insert(1,4)
>>> print(list)
[1, 4, 2, 3]
```

리스트 메서드

remove()
메서드

- 리스트 내 원하는 값을 삭제할 경우 사용, 특정값이 처음 나오는 위치의 값을 삭제

```
#.remove()
>>> list = [1, 2, 3, 4, 2, 5]
>>> list.remove(2)
>>> print(list)
[1, 3, 4, 2, 5]
```

count()
메서드

- 리스트 내 특정 요소의 개수를 세어서 반환

```
#.count()
>>> list = [1, 2, 3, 4, 1, 5, 1, 2]
>>> count = list.count(1)
>>> print(count)
3
```

리스트 메서드

pop()
메서드

- 리스트 내 특정 인덱스에 해당하는 요소를 제거하고 그 요소의 값을 반환
- 인덱스 미지정 시 마지막 원소를 제거

```
#.pop()
>>>numbers = [1, 2, 3, 4, 5]
>>>numbers.pop()
5

>>>print(numbers)
[1, 2, 3, 4]

>>>numbers.pop(1)
2

>>>print(numbers)
[1, 3, 4]
```

리스트 메서드

reverse()
메서드

- 리스트의 순서를 반대로 정렬, 원본 리스트를 직접 변경하므로 복사본을 추가로 생성하지 않아도 됨.

```
#.reverse()
>>> numbers = [1, 2, 3, 4, 5]
>>> print("Original list:", numbers)
Original list: [1, 2, 3, 4, 5]
>>> numbers.reverse()
>>> print("Reversed list:", numbers)
Reversed list: [5, 4, 3, 2, 1]
```

extend()
메서드

- 기존 리스트에 추가 리스트를 더해주는 메서드

```
#.extend()
>>> list1 = [1, 2, 3, 4]
>>> list2 = [5, 6, 7, 8]
>>> list1.extend(list2)
>>> print(list1)
[1, 2, 3, 4, 5, 6, 7, 8]
```

리스트 메서드

index()
메서드

- 리스트 내 특정 요소의 인덱스를 반환.
- 인덱스 미 지정 시 첫번째 요소의 인덱스를 반환.
- 찾고자 하는 요소가 없을 시, ValueError 예외가 발생하고, try-except구문을 통해 예외 처리가 가능.

```
#.index()
>>> fruits = ['apple', 'banana', 'cherry', 'banana']
>>> index = fruits.index('banana')
>>> print(index)
1

>>> fruits = ['apple', 'banana', 'cherry']
>>> try:
>>> index = fruits.index('grape')
>>> except ValueError:
>>> index = -1
>>> print(index)
-1
```

지금은?

튜플(Tuple)

튜플

- Python에서 자료형 중 하나이며, 리스트와 거의 동일하지만 설정 된 Data는 변경할 수 없음(상수(constant)와 같은 성질).
- 튜플은 리스트와 같이 여러 개의 요소(Element)를 가질 수 있는 순차적인 자료 구조이며, 소괄호()를 이용해서 생성
- 각 요소는 순서가 있고, 인덱스(Index)를 가지며, 첫번째 요소의 인덱스는 0부터 시작.


```
#튜플의 기본 생성방법
>>>numbers = (1, 2, 3, 4, 5)

#다형 자료 생성방법
>>>mixed_list = (1, “Python”, 3.14, True)

#tuple()함수를 이용한 생성
>>>str1 = “Python”
>>>tuple1 = tuple(str1)
>>>print(tuple1)
Python
```

리스트를 이용한 튜플 생성

리스트 이용
튜플 생성

- 튜플은 만들어 놓은 리스트를 이용해 튜플을 생성 가능.
- tuple()함수 또는 소괄호()안에 리스트 값을 넣어 생성.

list1	1 (Index : 0)	2 (Index : 1)	3 (Index : 2)	4 (Index : 3)	5 (Index : 4)

tuple(list),(list)

tuple1	1 (Index : 0)	2 (Index : 1)	3 (Index : 2)	4 (Index : 3)	5 (Index : 4)

```
#리스트를 이용하여 튜플의 생성방법
>>> list = [1, 2, 3, 4, 5]
>>> t = tuple(list)
>>> print(t)
(1, 2, 3, 4, 5)

>>> list = [1, 2, 3, 4, 5]
>>> t = (list)
>>> print(t)
(1, 2, 3, 4, 5)
```

튜플 인덱싱 & 슬라이싱

튜플
인덱싱

- 튜플의 인덱싱은 리스트와 마찬가지 방법으로 튜플명[인덱스]로 호출.

```
#튜플의 인덱싱
>>>tuple1 = (1, 2, 3)
>>>print(tuple1[0]) # 1
>>>print(tuple1[1]) # 2
>>>print(tuple1[2]) # 3
```

튜플
슬라이싱

- 튜플의 슬라이싱은 리스트와 마찬가지 방법으로 튜플명[start:end:step]으로 호출

```
#튜플의 슬라이싱
>>>tuple1 = (1, 2, 3, 4, 5)
>>>print(tuple1[1:3]) # (2, 3)
>>>print(tuple1[:3]) # (1, 2, 3)
>>>print(tuple1[1:]) # (2, 3, 4, 5)
>>>print(tuple1[::-1]) # (5, 4, 3, 2, 1)
```

튜플 연산

튜플 더하기 연산

- 튜플의 더하기는 두 개의 튜플을 연결하여 새로운 튜플을 생성.

```
#튜플의 더하기 연산
>>> t1 = (1, 2, 3)
>>> t2 = (4, 5, 6)
>>> t3 = t1 + t2
>>> print(t3)
(1, 2, 3, 4, 5, 6)
```

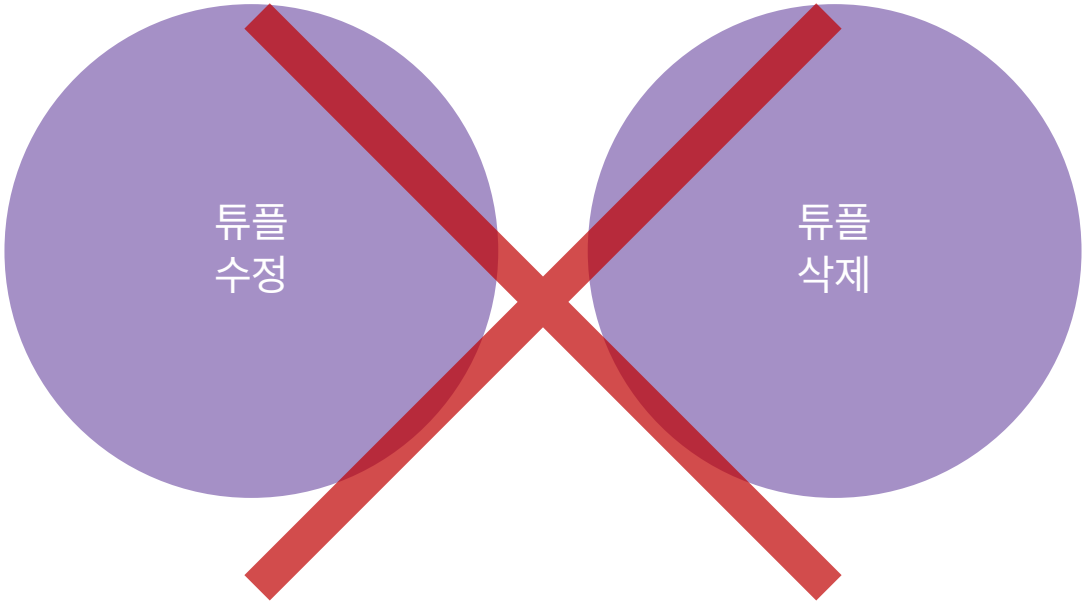
튜플 곱하기 연산

- 튜플의 곱하기는 기존의 튜플을 반복하여 새로운 튜플을 생성.

```
#튜플의 곱하기 연산
>>> t = (1, 2, 3)
>>> t2 = t * 2
>>> print(t2)
(1, 2, 3, 1, 2, 3)
```

튜플 연산

- 튜플은 요소의 내용을 변경하지 못하는 상수의 성질을 가지므로 수정 및 삭제 불가능.



튜플의 함수 및 메서드

len()
max()
min()

- 튜플의 기본 함수는 리스트와 거의 유사한 형태로 실행.

```
#튜플의 기본 함수
>>> t1 = (1, 2, 3, 4, 5)
>>> print(len(t1)) #요소의 개수 값
>>> print(max(t1)) #요소들 중 최대값
>>> print(min(t1)) #요소들 중 최소값
```

튜플은 변경 불가 자료형
메서드를 가질 수 없음

list()
tuple()

- list()함수와 tuple()함수는 튜플과 리스트 사이를 변환하는 함수.

```
#list(), tuple()함수
>>> t = (1, 2, 3)
>>> list(t)
[1, 2, 3]
>>> l = [1, 2, 3]
>>> tuple(l)
(1, 2, 3)
```

지금은?

딕셔너리(Dictionary)

딕셔너리

- Python에서 자료형 중 하나로 리스트와 유사하지만 딕셔너리의 경우 인덱스 대신 키와 값을 쌍으로 구성하여 Data를 저장.
- 딕셔너리는 중괄호({})를 사용하여 표현, 키와 값은 콜론(:)으로 구분.
- 리스트와는 다르게 순서가 없고, 튜플과 다르게 변경이 가능.
- 딕셔너리는 리스트와 달리 많은 양의 데이터를 효율적으로 저장하기에 적합

리스트(List)	
인덱스(Index)	변수(Value)
0	Python
1	1991
2	Guido van Rossum
3	Easy
4	72

딕셔너리(Dictionary)	
키(Key)	변수(Value)
Language	Python
Year	1991
Developer	Guido van Rossum
Level	Easy
Time	72

딕셔너리 생성 방법

중괄호({}) 사용 생성

- 기본적인 딕셔너리 생성방법으로 {}를 이용 생성.

```
#딕셔너리 생성(문자키)
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul ' }

#딕셔너리 생성2(숫자키)
>>> numbers = {1: 'one', 2: 'two', 3: 'three'}
```

dict()함수 사용 생성

- dict()함수는 딕셔너리 생성 함수로 키와 값의 쌍으로 구성.

```
#dict()함수를 통한 딕셔너리 생성
>>> person = dict(name='John Doe', age=30, city='Seoul')
>>> print(person)
{'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
```

딕셔너리 추가 & 수정

요소 추가

- “딕셔너리 이름[Key] = value”의 형태로 새로운 요소 추가.

```
#딕셔너리 요소 추가
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> person['gender'] = 'Male'
>>> print(person)
{'name': 'John Doe', 'age': 30, 'city': 'Seoul', 'gender': 'Male'}
```

요소 변경

- “딕셔너리 이름[Key] = value”의 형태로 요소의 값 변경.

```
#딕셔너리 요소 변경
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> person['age'] = 31
>>> print(person)
{'name': 'John Doe', 'age': 31, 'city': 'Seoul'}
```

딕셔너리 요소 호출

Key 호출

- “딕셔너리 이름[Key]”을 통해 요소를 호출

```
#key를 이용 호출
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> print(person['name'])
'John Doe'
```

get()메서드 호출

- get() 메서드를 통해 요소를 호출하고, 값이 없을 경우 None이 반한.

```
#get()메서드 호출
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> print(person.get('gender'))
None
```

딕셔너리 요소 삭제

pop() 메서드

- pop()메서드를 활용해서 삭제하고자 하는 Key값을 호출해서 삭제.

```
#pop()를 이용 삭제
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> person.pop('age')
>>> print(person)
{'name': 'John Doe', 'city': 'Seoul'}
```

del 문법

- get() 메서드를 통해 요소를 호출하고, 값이 없을 경우 None이 반한.

```
#del문법을 이용 삭제
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> del person['age']
>>> print(person)
{'name': 'John Doe', 'city': 'Seoul'}
```

딕셔너리 요소 삭제

clear() 메서드

- clear()메서드를 활용해서 모든 요소를 삭제.

```
#clear()를 이용 전체 삭제
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> person.clear()
>>> print(person)
{}
```

딕셔너리 메서드

keys() 메서드

- keys()메서드를 활용하면 딕셔너리의 키(key)값만 호출.

```
#keys()메서드
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> print(person.keys())
dict_keys(['name', 'age', 'city'])
```

value() 메서드

- value()메서드를 활용하면 디셔너리의 값(value)만 호출.

```
#value()메서드
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> print(person.values())
dict_values(['John Doe', 30, 'Seoul'])
```

딕셔너리 메서드

copy() 메서드

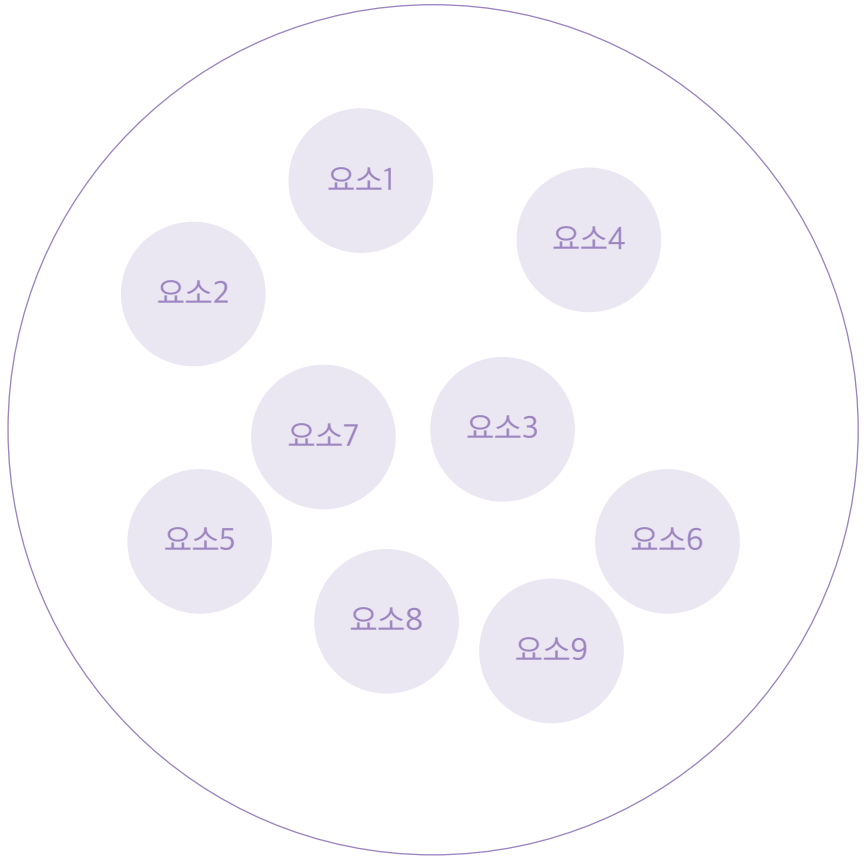
- copy()메서드를 활용하면 딕셔너리의 복사본 생성.

```
#copy()메서드
>>> person = {'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
>>> person_copy = person.copy()
>>> print(person_copy)
{'name': 'John Doe', 'age': 30, 'city': 'Seoul'}
```

집합(Set)

- Python에서 집합은 고유한 요소의 모음으로 단일 변수에 여러 항목을 저장.
- 요소의 순서가 없어 인덱스로 접근이 불가.
- 중복은 허용되지 않음.
- 변경 불가능한 자료형만 사용 가능.

집합(Set)



집합 생성 방법

set() 함수

- set()함수를 활용하여 변경 가능한 자료형(리스트, 딕셔너리, 스트링 등)을 집합으로 변환하거나 생성.

```
#set()함수 활용 집합 생성
>>> set_example = set([1, 2, 3, 4, 5])
>>> print(set_example)
{1, 2, 3, 4, 5}
```

직접 값 할당

- 변수에 직접 값을 할당하여 집합 생성.

```
#직접 값 할당해서 집합 생성
>>> set_example = {1, 2, 3, 4, 5}
>>> print(set_example)
{1, 2, 3, 4, 5}
```

집합 요소 호출

반복문

- 반복문을 이용하여 집합의 요소 호출.

```
#반복문을 이용하여 요소 호출
>>> set_example = {1, 2, 3}
>>> for i in set_example:
>>> print(i)
1
2
3
```

In연산자

- 집합에 특정 요소가 있는 지 in연산자를 이용해서 확인, True / False값을 반환.

```
#in연산자를 이용해서 요소 확인
>>> fruits = {"apple", "banana", "cherry"}
>>> print("apple" in fruits)
>>> print("grape" in fruits)
True
False
```

집합 요소 추가

add() 메서드

- add()메서드는 집합에 단일 요소를 추가하는 데 사용.

```
#add() 메서드
>>> set_example = {1, 2, 3, 4, 5}
>>> set_example.add(6)
>>> print(set_example)
{1, 2, 3, 4, 5, 6}
```

update() 메서드

- update()메서드는 집합에 여러 요소를 추가하는 데 사용.

```
#update() 메서드
>>> set_example = {1, 2, 3, 4, 5}
>>> set_example.update([6, 7, 8])
>>> print(set_example)
{1, 2, 3, 4, 5, 6, 7, 8}
```

집합 요소 제거

remove() 메서드

- remove()메서드는 집합에서 특정 요소를 제거하는데 사용, 요소가 집합에 없을 경우, KeyError가 발생.

```
#remove() 메서드
>>> set_example = {1, 2, 3, 4, 5}
>>> set_example.remove(5)
>>> print(set_example)
{1, 2, 3, 4}
```

discard() 메서드

- discard()메서드는 집합에서 특정 요소를 제거하는데 사용, 요소가 집합에 없을 경우 KeyError가 발생 않함.

```
#discard() 메서드
>>> set_example = {1, 2, 3, 4, 5}
>>> set_example.discard(6)
>>> print(set_example)
{1, 2, 3, 4, 5}
```

집합의 연산

- 두 집합의 합집합은 두 집합의 모든 요소를 포함하는 집합으로 union() 메서드 또는 | 연산자를 사용.

```
>>> set_a = {1, 2, 3, 4, 5}
>>> set_b = {4, 5, 6, 7, 8}
>>> set_c = set_a.union(set_b)
>>> print(set_c)
{1, 2, 3, 4, 5, 6, 7, 8}
>>> set_c = set_a | set_b
>>> print(set_c)
{1, 2, 3, 4, 5, 6, 7, 8}
```

- 두 집합의 교집합은 두 집합의 공통 요소만 포함하는 집합으로 intersection()메서드 또는 & 연산자를 사용.

```
>>> set_a = {1, 2, 3, 4, 5}
>>> set_b = {4, 5, 6, 7, 8}
>>> set_c = set_a.intersection(set_b)
>>> print(set_c)
{4, 5}
>>> set_c = set_a & set_b
>>> print(set_c)
{4, 5}
```

집합의 연산

차집합

- 두 집합의 차집합은 한 집합에는 존재하고, 다른 집합에 없는 요소를 포함하는 집합, difference()메서드 또는 - 연산자 사용.

```
>>> set_a = {1, 2, 3, 4, 5}
>>> set_b = {4, 5, 6, 7, 8}
>>> set_c = set_a.difference(set_b)
>>> print(set_c)
{1, 2, 3}
>>> set_c = set_a - set_b
>>> print(set_c)
{1, 2, 3}
```

대칭 차집합

- 두 집합의 대칭 차집합은 공통된 요소만 제외하고 모든 요소를 포함하는 집합, symmetric_difference()메서드 또는 ^ 연산자 사용.

```
>>> set_a = {1, 2, 3, 4, 5}
>>> set_b = {4, 5, 6, 7, 8}
>>> set_c = set_a.symmetric_difference(set_b)
>>> print(set_c)
{1, 2, 3, 6, 7, 8}
>>> set_c = set_a ^ set_b
>>> print(set_c)
{1, 2, 3, 6, 7, 8}
```

집합의 메서드

set()
& frozenset()

- Set()함수는 가변 집합을 생성하고, frozenset()함수는 불변 집합을 생성.
- Frozenset()은 불변 집합이기 때문에 add()메서드로 요소를 추가하면 AttributeError이 발생.

```
>>>set_example = set([1, 2, 3, 4, 5])
>>>set_example.add(6)
>>>print(set_example)
{1, 2, 3, 4, 5, 6}

>>>frozenset_example = frozenset([1, 2, 3, 4, 5])
>>>frozenset_example.add(6)
>>>AttributeError: 'frozenset' object has no attribute 'add'
```

지금은?

조건문

조건문

- 조건문이란 특정 조건이 참(True)인지 거짓(False)인지를 판단하여 결과에 따라 코드 블록을 수행하는 구조.
- 파이썬 에서는 3가지 조건문 활용(if문 / if-else문 / if-elif-else문).
- If문은 조건식이 참인 경우에만 코드 블록이 수행.
- If-else문은 조건식이 참인 경우에는 코드 블록 1이 수행되고 거짓인 경우 코드 블록 2가 수행.
- If-elif-else문은 여러 조건식을 판단, 각 조건식이 참인 경우 해당 코드 블록 수행, 어떤 조건식도 참이 아닌 경우 else 블록 수행.

```
# if문 기본 구조
>>>if 조건식 :
            코드블록

#if-else문 기본 구조
>>>if 조건식 :
            코드블록1
>>>else:
            코드블록 2

#if-elif-else문 기본구조
>>>if 조건식 1:
            코드블록1
>>>elif 조건식2:
            코드블록2
>>>else:
            코드블록 n
```

조건문

if문

- If문은 조건식이 참일 경우에만 다음 블록의 코드 수행.

```
# if문 기본 구조
x = 5
if x > 0:
print("양수입니다")
```

if-else문

- If-else문은 조건식이 참일 경우와 거짓일 경우 각각 다른 코드 블록을 실행.

```
# if-else문 기본 구조
x = 5
if x > 0:
print("양수입니다")
else:
print("음수입니다")
```

조건문

if-elif-else문

- If-elif-else문은 여러 조건식을 판단하여 각각 다른 코드 블록을 수행

```
# if-elif-else문 기본 구조
x = 0
if x > 0:
print("양수입니다")
elif x == 0:
print("0입니다")
else:
print("음수입니다")
```

조건문 연산자

In 연산자

- “in”연산자는 특장 값이 컬렉션 자료형(리스트, 튜플 등)에 포함되어 있는지 확인할 경우 사용.

```
# If문 “in”연산자
fruits = ["apple", "banana", "cherry"]
if "apple" in fruits:
print("apple is in the fruits list")
else:
print("apple is not in the fruits list")
```

- “in”연산자는 if-elif-else문에서도 사용 가능

```
# If-elif-else문 “in”연산자
numbers = [1, 2, 3, 4, 5]
if 6 in numbers:
print("6 is in the numbers list")
elif 7 in numbers:
print("7 is in the numbers list")
else:
print("6 and 7 are not in the numbers list")
```

조건문 연산자

and 연산자

- “and”연산자는 두 개의 조건이 모두 True일 때, True를 반환

```
>>> a = 10
>>> b = 20
>>> c = 30
>>> print(a < b and b < c)
True
>>> print(a > b and b < c)
False
```

or 연산자

- “or”연산자는 두 개의 조건 중 하나라도 True이면, True를 반환(두 조건 모두 False일 때, False 반환).

```
>>> a = 10
>>> b = 20
>>> c = 30
>>> print(a < b or b < c)
True
>>> print(a > b or b < c)
True
>>> print(a > b or b > c)
False
```

조건문 연산자

‘and’ + ’or’ 연산자 연결 사용

- “and”와 “or " 연산자는 여러 개를 연결하여 사용 가능.

```
a = 10
b = 20
c = 30
if a < b and b < c or a > c:
print("a is less than b and b is less than c or a is greater than c")
else:
print("none of the conditions are met")
```

괄호() 사용

- 괄호를 사용하면 표현식을 그룹핑하거나 우선 순위를 변경.

```
a = 10
b = 20
c = 30
if (a < b and b < c) or (a > c):
print("a is less than b and b is less than c or a is greater than c")
else:
print("none of the conditions are met")
```

반복문

반복문

- 반복문이란 특정 코드 블록을 반복하여 수행하는 문법.
- 파이썬 에서는 for 반복문과 while 반복문 사용.
- For 반복문은 특정한 범위 내에서 일정한 패턴에 따라 반복적으로 코드를 실행하는 구조.
- While 반복문은 조건식이 참인 동안 특정 코드 블록을 반복하여 실행하는 구조.

```
#for 반복문 기본 구조
>>>for 변수 in 범위 :
                코드블록

#while 반복문 기본 구조
>>>while 조건식 :
                코드블록
```

반복문

For 반복문
(range())

- Range()함수는 일정 범위의 연속된 정수를 생성하는 데 사용.
- 일정 횟수만큼 반복 수행하거나, 특정 범위의 정수를 이용한 작업을 수행할 때 사용.

```
#range를 이용한 반복문
>>> for i in range(1, 11):
>>> print(i)

#리스트를 이용한 반복문
>>> fruits = ["사과", "바나나", "포도"]
>>> for fruit in fruits:
>>> print(fruit)

#문자열을 이용한 반복문
>>> text = "파이썬"
>>> for character in text:
>>> print(character)
```

반복문

For 반복문
(enumerate())

- enumerate()함수는 리스트와 인덱스 값을 동시에 순회.

```
#enumerate()를 이용한 반복문
>>> names = ["철수", "영희", "민수", "수지"]
>>> for idx, name in enumerate(names):
>>> print(idx, name)
```

중첩 반복문

For문 중첩

- 중첩 반목문은 하나의 반복문 안에 또 다른 반복문이 포함된 경우.
- 이차원 배열이나 행렬과 같은 다차원 데이터 처리시 사용
- For문과 While문 모두에서 사용 가능

```
>>> for 변수1 in 반복 가능한 객체1:
>>> for 변수2 in 반복 가능한 객체2:
>>> 실행할 코드
```

```
>>> for i in range(2, 10):
>>> for j in range(1, 10):
>>> print(f"{i} * {j} = {i * j}")
>>> print()
```

중첩 반복문

While문 중첩

- 중첩 반목문은 하나의 반복문 안에 또 다른 반복문이 포함된 경우.
- 이차원 배열이나 행렬과 같은 다차원 데이터 처리시 사용
- For문과 While문 모두에서 사용 가능

```
>>> while 조건1:
>>> while 조건2:
>>> 실행할 코드
>>> 조건1에 관련된 변수 변경
```

```
>>> i = 2
>>> while i < 10:
>>> j = 1
>>> while j < 10:
>>> print(f"{i} * {j} = {i * j}")
>>> j += 1
>>> print()
>>> i += 1
```

중첩 반복문

For문과
While문 중첩

- 중첩 반목문은 하나의 반복문 안에 또 다른 반복문이 포함된 경우.
- 이차원 배열이나 행렬과 같은 다차원 데이터 처리시 사용
- For문과 While문 모두에서 사용 가능

```
>>> for 변수 in 반복 가능한 객체:
>>> while 조건:
>>> 실행할 코드
>>> 조건에 관련된 변수 변경
>>> while 조건:
>>> for 변수 in 반복 가능한 객체:
>>> 실행할 코드
>>> 조건에 관련된 변수 변경
```

제어문

제어문

- 반복문에는 break, continue, pass의 세가지 제어문 사용.
- 제어문은 반복문의 실행 흐름을 제어하는 데 사용.

```
#Break제어문 / 현재 실행 중인 반복문을 즉시 종료하고 다음 코드로 이동.
>>> n = 1
>>> while True:
>>> if n > 10:
>>> break
>>> print(n)
>>> n += 1

#Continue제어문 / 현재 실행 중인 반복문의 나머지 부분을 건너뛰고 다음 반복으로 진행.
>>> for i in range(1, 11):
>>> if i % 2 == 0:
>>> continue
>>> print(i)

#Pass제어문 / pass문은 별도 동작을 하지 않고, 구조를 잡ㅓ나 구현되지 않은 부분을 표시하는데 사용.
>>> for i in range(1, 11):
>>> if i % 2 == 0:
>>> pass
>>> else:
>>> print(i)
```


Q & A

Quiz

1번 어떤 수가 양수인지 음수인지 판단하여, 양수일 경우 "양수입니다"라는 메시지를, 음수일 경우 "음수입니다"라는 메시지를 출력하는 프로그램을 작성하세요.

2번 어떤 수가 3의 배수인지 판단하여, 3의 배수일 경우 "3의 배수입니다"라는 메시지를, 아닐 경우 "3의 배수가 아닙니다"라는 메시지를 출력하는 프로그램을 작성하세요.

3번 어떤 수가 가장 큰 수인지 판단하여, 가장 큰 수일 경우 "가장 큰 수입니다"라는 메시지를, 아닐 경우 "가장 큰 수가 아닙니다"라는 메시지를 출력하는 프로그램을 작성하세요.

4번 1부터 100까지의 숫자 중 5의 배수의 합을 구하세요.

5번 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 리스트에서 최대값, 최소값, 평균을 구하세요.

Quiz1

1번

어떤 수가 양수인지 음수인지 판단하여, 양수일 경우 "양수입니다"라는 메시지를, 음수일 경우 "음수입니다"라는 메시지를 출력하는 프로그램을 작성하세요.

```
num = int(input("수를 입력하세요: ")) #변수를 입력받습니다.
if num > 0: #입력받은 변수가 0보다 큰지 판단하고, 클 경우 양수로 판단합니다.
    print("양수입니다")
elif num < 0: #입력받은 변수가 0보다 작은지 판단하고, 작을 경우 음수로 판단합니다.
    print("음수입니다")
else: #그 외 경우는 0으로 판단합니다.
    print("0입니다")
```

Quiz2

2번

어떤 수가 3의 배수인지 판단하여, 3의 배수일 경우 "3의 배수입니다"라는 메시지를, 아닐 경우 "3의 배수가 아닙니다"라는 메시지를 출력하는 프로그램을 작성하세요.

```
n = int(input("수를 입력하세요: ")) #변수를 입력 받습니다.
if n % 3 == 0: #입력받은 변수를 3으로 나눈 후 나머지 값이 0이면 3의 배수로 표기합니다.
    print("3의 배수입니다.")
else: #그 외의 경우는 3의 배수가 아니라고 표기합니다.
    print("3의 배수가 아닙니다.")
```

Quiz3

3번

어떤 수가 가장 큰 수인지 판단하여, 가장 큰 수일 경우 "가장 큰 수입니다"라는 메시지를, 아닐 경우 "가장 큰 수가 아닙니다"라는 메시지를 출력하는 프로그램을 작성하세요.

```
숫자1 = int(input("첫 번째 숫자를 입력하세요: "))#첫번째 변수를 입력 받습니다.
숫자2 = int(input("두 번째 숫자를 입력하세요: "))#두번째 변수를 입력 받습니다.
숫자3 = int(input("세 번째 숫자를 입력하세요: "))#세번째 변수를 입력 받습니다.

가장_큰_수 = max(숫자1, 숫자2, 숫자3)#max()함수를 이용해서 입력 받은 숫자 중 가장 큰 수를 가장_큰_수 변수에 담아줍니다.

if 숫자1 == 가장_큰_수: #가장_큰_수에 담긴 값이 숫자1이면 표기합니다.
    print("첫 번째 숫자는 가장 큰 수입니다.")
elif 숫자2 == 가장_큰_수: #가장_큰_수에 담긴 값이 숫자2이면 표기합니다.
    print(“두 번째 숫자는 가장 큰 수입니다.”)
Else: #가장_큰_수에 담긴 값이 숫자3이면 표기합니다.
    print("세 번째 숫자는 가장 큰 수입니다.")
```

Quiz4

4번

1부터 100까지의 숫자 중 5의 배수의 합을 구하세요.

```
sum = 0 #초기값을 0으로 합계값 sum을 할당합니다.
for i in range(1,101): #1~101까지의 값을 i변후로 반복 입력 합니다.
if i % 5 == 0: #입력받은 변수를 5로 나눈 뒤 나머지가 0이면 sum변수에 할당합니다.
sum += i #할당받은 변수가 해당하면 이전 값에 더해줍니다.
print(sum)#sum값을 출력해줍니다.
```

Quiz5

5번

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 리스트에서 최대값, 최소값, 평균을 구하세요.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]#1~10까지 수를 변수로 할당합니다.

max_num = numbers[0]#변수로 입력 받은 수들 중 최대값을 찾습니다.
min_num = numbers[0]#변수로 입력 받은 수들 중 최소값을 찾습니다.
sum = 0 #sum변수에는 0을 할당합니다.

for num in numbers:#변수로 입력받은 numbers의 요소들을 반복문으로 대입합니다.
  if num > max_num:#입력 받은 변수가 최대값보다 클 경우 최대값에 대입합니다.
    max_num = num
  if num < min_num:#입력 받은 변수가 최소값보다 작을 경우 최소값에 대입합니다.
    min_num = num
    sum += num #변수들의 합을 구합니다.

average = sum / len(numbers)#변수들의 합을 변수의 수를 카운트한 값으로 나눠 평균을 구합니다.

print("최대값:", max_num)#최대값을 출력합니다.
print("최소값:", min_num)#최소값을 출력합니다.
print("평균:", average)#평균값을 출력합니다.
```

강 의 일 정

IoT 기반 공정설비모니터링 프로그래밍(5d)　　로봇 프로그래밍(AI+파이썬)(3d)

1회 차 8월 19일(월)	IoT 기반 공정설비 모니터링의 이해	IoT의 이해 / 센서 & 모니터링 시스템 / IoT네트워크	이론
2회 차 8월 20일(화)	라즈베리파이 & 센서 프로그래밍 1	Rpi의 이해 / Rpi 개발환경 구축 / 센서 프로그래밍(LED/온·습도/초음파…)	이론 /실습
3회 차 8월 21일(수)	라즈베리파이 & 센서 프로그래밍 2	Rpi 원격 개발환경 / 센서 응용 프로그래밍	이론 /실습
4회 차 8월 22일(목)	Node-Red 제어&모니터링 프로그래밍 1	Node-Red의 이해 / 개발환경 구축 / 센서 모니터링 및 제어	이론 /실습
5회 차 8월 26일(월)	Node-Red 제어&모니터링 프로그래밍 2	Dashboard 설계 / http통신 & DB / MQTT	이론 /실습
6회 차 8월 27일(화)	ESP32 마이크로 파이썬 로봇 프로그래밍 1	ESP32의 이해 / 마이크로 파이썬 개발환경 구축 / 로봇 프로그래밍(LEC/온·습도/초음파…)	이론 /실습
7회 차 8월 28일(수)	ESP32 마이크로 파이썬 로봇 프로그래밍 2	로봇 프로그래밍(DC모터/서보모터/LCD…) / 웹 서버 구축 및 제어 / AI 프로그래밍 이해	이론 /실습
8회 차 9월 4일(수)	프로젝트 & 평가	팀 프로젝트 및 평가	실습 /평가