D&A

Basic Session 2차시

파이썬 기초 1.

2022 / 03 / 15

D&A 운영진 이경욱

CONTENTS.

01 – 자료형

언어의 기본형태

01. 자료형 – 숫자형과 문자열

숫자형 – Numeric

```
a = 7      # 정수형 int
b = 0.34   # 실수형 float
```

```
type(7)
```

```
int
```

➡ ** type()으로 자료형 확인**

숫자형 - 연산

```
a + b
```

```
7.34
```

```
a - b
```

```
6.66
```

```
a / b
```

```
20.588235294117645
```

```
a * b
```

```
2.3800000000000003
```

➡ 사칙연산!

문자열 - string

```
a = "파이썬..너 누구냐"
b = "AI빅데이터"
c = "융합경영학과"
```

-> (")또는 (')로 양쪽 둘러싸기

```
a = "It ain't cool"
a
```

```
"It ain't cool"
```

➡ 문자열 내에 (')를 포함시키고 싶으면
문자열을 (")로 감싸기 (그 반대도)

문자열 - 연산

```
b + c
```

```
'AI빅데이터융합경영학과'
```

➡ 문자열끼리 덧셈

```
a * 2
```

```
"파이썬..is..'python'파이썬..is..'python'"
```

➡ 곱셈만큼 반복

01. 자료형 – 인덱싱과 슬라이싱(feat. 문자열)

인덱싱 – 인덱스(위치정보)로 해당 위치의 요소 반환

```
a = "인덱싱예제입니다"
a[0]
```

```
'인'
```

➡ 해당 위치를 0부터 세주고 []에 넣어 요소 반환

```
a = "인덱싱예제입니다"
a[-1]
```

```
'다'
```

➡ 거꾸로는 -1부터 세주기!

슬라이싱 – 인덱스값으로 자르기

```
a = "인덱싱예제입니다"
a[0:3]
```

```
'인덱싱'
```

➡ 끝 인덱스는 포함시키지 않음

슬라이싱

```
a = "인덱싱예제입니다"
a[:]
```

```
'인덱싱예제입니다'
```

➡ 전체를 반환

```
a = "인덱싱예제입니다"
a[:-1]
```

```
'인덱싱예제입니'
```

➡ 거꾸로 인덱싱 적용시

```
a = "인덱싱예제입니다"
a[::-1]
```

```
'다니입제예싱덱인'
```

➡ 슬라이싱을 활용해 문자열 거꾸로 바꾸기

01. 자료형 – bool 불 자료형

bool

```
a = True
b = False
type(a)
```

```
bool
```

➡ True 또는 False 즉 참과 거짓
두 가지 값을 가짐 (첫글자는 대문자로!)

두 값의 비교 결과로의 bool자료형

```
2 > 1
```

```
True
```

```
2 < 1
```

```
False
```

➡ 설정한 조건이 참인지 거짓인지 반환

자료형의 참과 거짓

```
bool(0)
```

```
False
```

```
bool(1)
```

```
True
```

```
bool(1345)
```

```
True
```

➡ 숫자형이 0 -> False
숫자형이 0이 아님 -> True

```
a = []
b = [1, 2, 3]
bool(a)
```

```
False
```

```
bool(b)
```

```
True
```

➡ 자료형이 비어있다 -> False
자료형이 비어있지 않다 -> True

01. 자료형 – 날짜형(feat. datetime)

datetime으로 날짜형 만들기

```
import datetime

day1 = datetime.date(2019, 1, 14)
datetime1 = datetime.datetime(2019, 1, 14, 14, 0, 0)
time1 = datetime.time(17, 30, 30)

print(day1)
print(datetime1)
print(time1)
```

```
2019-01-14
2019-01-14 14:00:00
17:30:30
```

➡ Import datetime을 통해 날짜형 자료 생성을 위한 모듈 가져오기!

➡ datetime.date로 날짜 생성

➡ datetime.datetime으로 날짜 + 시간 생성

➡ datetime.time으로 시간 생성

datetime 속성 가져오기

```
print(datetime1.year)
print(datetime1.month)
print(datetime1.day)
print(datetime1.hour)
print(datetime1.minute)
print(datetime1.second)
print(datetime1.weekday())
```

```
2019
1
14
14
0
0
0
```

➡ 년, 월, 일, 시, 분, 초, 요일

➡ 요일의 경우 월요일이 0부터 시작

➡ datetime.date의 경우 시, 분, 초 제외

➡ datetime.time의 경우 시, 분, 초만

01. 자료형 – 날짜형(feat. datetime)

두 date의 차이

```
day1 = datetime.date(2019, 1, 14)
day2 = datetime.date(2020, 8, 21)

diff = day2 - day1
diff
```

```
datetime.timedelta(days=585)
```

➡ 단순 (-) 연산으로 두 날짜의 차이가 datetime.timedelta 객체로 반환

➡ datetime.date로 날짜 생성

➡ datetime.datetime으로 시간까지 생성

두 datetime의 차이

```
datetime1 = datetime.datetime(2019, 1, 14, 14, 0, 0)
datetime2 = datetime.datetime(2019, 1, 25, 15, 0, 0)

datetime_diff = datetime2 - datetime1
datetime_diff
```

```
datetime.timedelta(days=11, seconds=3600)
```

➡ 단순 (-) 연산으로 두 날짜의 차이가 timedelta 객체로 반환

➡ day + seconds의 차이가 반영됨

01. 자료형 – 날짜형 (feat.Pandas)

Pandas 모듈을 사용한 날짜형(pandas -> pd)

```
import pandas as pd

now = pd.to_datetime("now") # UTC(국제 표준시)기준
today = pd.to_datetime("today") # 우리나라 기준
print(now)
print(today)
```

```
2022-03-01 14:30:42.335329
2022-03-01 23:30:42.335329
```

```
a = pd.to_datetime('2020-03-16 10:20:30', format = '%Y-%m-%d %H:%M:%S')
b = pd.to_datetime('03162020 10:20:30',format='%m%d%Y %H:%M:%S')
c = pd.to_datetime("2020/03/16 10:20:30", format = "%Y/%m/%d %H:%M:%S")

print(a)
print(b)
print(c)
```

```
2020-03-16 10:20:30
2020-03-16 10:20:30
2020-03-16 10:20:30
```

날짜형 객체 만들기

시간 형식 지정자	의미	결과
%a	요일 (짧은 이름)	Sun, Mon, ..., Sat
%A	요일 (긴 이름)	Sunday, Monday...
%w	요일 (숫자, 0부터 일요일)	0, 1, 2, 3, 4, 5, 6
%d	날짜 (2자리)	01, 02, ..., 31
%b	월 (짧은 이름)	Jan, Feb, ..., Dec
%B	월 (긴 이름)	January, Feburary,...
%m	월 (숫자)	01, 02, ..., 12
%y	연 (2자리)	00, 01, ..., 99
%Y	연 (4자리)	1960, 2002, 2020, ...
%H	시간 (24시간)	00, 01, ..., 23
%I	시간 (12시간)	01, 02, ..., 12
%M	분 (2자리)	00, 01, ..., 59
%S	초 (2자리)	00, 01, ..., 59

➡ 해당하는 지정자를 사용해 문자열의 형태를
format = " " 에 입력!

01. 자료형 – List 리스트

List

```
a = []                                  # 빈 리스트
b = [1, 2, 3]                           # 숫자형
c = ['Life', 'is', 'too', 'short'] # 문자열
d = [1, 2, 'Life', 'is']                # 숫자형과 문자열
e = [1, 2, ['Life', 'is']]              # 리스트 내 리스트
```

➡ 리스트의 요소는 , (쉼표)로 구분

➡ 리스트의 요소로 어떠한 자료형도 포함시킬 수 있음

List 인덱싱 – 인덱스값(위치정보)로 요소 반환

```
a = [1, 2, 3, 4]
a[0]
```

```
1
```

➡ 요소의 위치를 0부터 세어 []안에 넣어 요소 반환

```
a = [1, 2, ['Life', 'is']]
a[2][0]
```

```
'Life'
```

➡ 이중 리스트 인덱싱

List 슬라이싱 – 인덱스값으로 리스트를 자름

```
a = [1, 2, 3, 4]
a[0:2]
```

```
[1, 2]
```

➡ 인덱스값을 이용 끝단위를 포함X

```
a = [1, 2, 3, 4]
a[::-1]
```

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

➡ 문자열과 동일

List 연산

```
a = [1, 2, 3]
b = [4, 5, 6]
a + b
```

```
[1, 2, 3, 4, 5, 6]
```

➡ 문자열과 동일한 덧셈, 곱셈

```
a * 3
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

01. 자료형 – tuple 튜플

tuple

```
t1 = ()
t2 = (1,)
t3 = (1, 2, 3)
t4 = 1, 2, 3
t5 = ('a', 'b', ('ab', 'cd'))
```

➡ 리스트와 비슷하나 몇가지 차이점 존재

1) 요소를 콤마로 구분하고 ()로 둘러싸기
2) 하나의 요소만 들어갈 땐 ,로 마무리
3) 괄호를 생략해도됨
4) ** 값의 생성과 삭제, 수정이 불가능함 **

```
t4 = 1, 2, 3
t4
```

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

➡ 만들 때만 괄호가 생략되고
만들어진 튜플은 괄호존재

튜플의 인덱싱과 슬라이싱

```
t3 = (1, 2, 3)
t3[1]
```

```
2
```

```
t3[0:1]
```

```
(1,)
```

```
t3[:]
```

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

➡ 리스트와 동일

01. 자료형 – dictionary 딕셔너리(사전)

dic

```
a = {"이름" : "이경욱", "나이" : 20, "학교" : "국민대"}
a
```

```
{'이름': '이경욱', '나이': 20, '학교': '국민대'}
```

➡ {key1 : value1, key2 : value2, …}의 형태

➡ Key와 대응관계에 있는 value를 나타내는 자료형

➡ Key에는 변하지 않는 값, value는 모두 사용 가능

dict 요소 반환하기 – 인덱싱 및 슬라이싱 미지원

```
a = {"이름" : "이경욱", "나이" : 20, "학교" : "국민대"}
a["이름"]
```

```
'이경욱'
```

➡ dict[해당 key] **key가 중복될 경우 1개를 제외한 나머지가 무시됨

dict 새로운 key:value 추가하기

```
a = {"이름" : "백종원"}
a["나이"] = 27
a
```

```
{'이름': '백종원', '나이': 27}
```

➡ dict[새로운 key] = 새로운 value

```
a = {[1, 2] : "dict"}
```

```
---------------------------------------------------------------
TypeError                                 Traceback (
~₩AppData₩Local₩Temp/ipykernel_11400/2852043892.py
----> 1 a = {[1, 2] : "dict"}

TypeError: unhashable type: 'list'
```

➡ Key는 변하지 않는 값을 사용해야함으로
List는 사용불가(그럼 tuple은??)

01. 자료형 – set 집합

set

```
a = {1, 1, 2, 3}
a
```

```
{1, 2, 3}
```

```
type(a)
```

```
set
```

➡ {}로 요소를 콤마,로 구분

➡ 중복을 허용하지 않음

➡ 순서가 없다 (인덱싱, 슬라이싱 미지원)

```
s1 = set([1,1, 2, 2, 2, 3])
s1
```

```
{1, 2, 3}
```

```
a = set("파파이이썬썬")
a
```

```
{'썬', '이', '파'}
```

➡ set() 함수에 리스트나 문자열을 넣어 set 자료형으로 만들기

➡ 리스트와 문자열의 중복된 요소가 제거된 set이 만들어짐

set의 유용한 활용

```
a = {1, 2, 3, 4}
b = {3, 4, 5, 6}
a & b
```

```
{3, 4}
```

➡ 교집합

```
a | b
```

```
{1, 2, 3, 4, 5, 6}
```

➡ 합집합

```
a - b
```

```
{1, 2}
```

➡ 차집합

01. 자료형을 담는 그릇, 변수

변수 할당

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

```
[1, 2, 3]
```

➡ 이제 a는 리스트를 가리키는 변수가 됨

변수명 설정 규칙

➡
1) 영문과 숫자로 이루어져야한다
2) 숫자부터 시작할 수 없다
3) 대소문자를 구분한다(ex. A와 a를 구분)
4) 키워드*를 사용하지 못함

키워드?

-> 파이썬에서 이미 각각의 용도로 사용하고 있는 단어

```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class',
'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import',
'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yiel
d']
```

같은 값에 대한 여러 변수

```
a = [1, 2, 3]
b = a
```

➡ a에 리스트 할당 후 b에 a를 할당

```
id(a)
```

```
2328622011328
```

```
id(b)
```

```
2328622011328
```

➡ Id()를 사용해 변수의 메모리 주소 확인결과 동일

➡ a를 수정, 삭제할 경우 b도 변경됨

```
a = [1, 2, 3]
b = a[:]
```

➡ 위 경우와 같은 값을 가리키지만, 메모리 주소가 다름

```
id(a)
```

```
2328622012224
```

```
id(b)
```

```
2328621966400
```

➡ a를 수정, 삭제해도 b에 영향없음

02 - 연산자

02. 연산자 – 산술연산자와 비교연산자

산술 연산자

```
a = 15
b = 6
a // b
```

```
2
```

```
a % b
```

```
3
```

```
a ** b
```

```
11390625
```

➡ 숫자형의 사칙연산 이외에 세가지 더 존재

1) // -> 몫을 반환
2) % -> 나머지를 반환
3) ** -> 제곱

비교 연산자

==	값이 동일하다
!=	값이 동일하지 않다
>	왼쪽 값이 오른쪽 값보다 크다
<	왼쪽 값이 오른쪽 값보다 작다
>=	왼쪽 값이 오른쪽 값보다 크거나 동일하다
<=	왼쪽 값이 오른쪽 값보다 작거나 동일하다

Ex)

```
354 > 4561
```

```
False
```

➡ 비교 조건이 참인 거짓인지 bool자료형 반환

02. 연산자 – 멤버연산자와 논리연산자

멤버 연산자

in	list 내에 포함되어 있으면 참
not in	list 내에 포함되어 있지 않으면 참

Ex)

```
a = 1
b = [1, 2, 3, 4]
c = [2, 3, 4, 5]
```

```
a in b
```

```
True
```

```
a not in c
```

```
True
```

```
a in c
```

```
False
```

a가 b에 포함되어 있는지를 bool형태로 반환

논리 연산자

and	논리 AND 연산. 둘다 참일때만 참
or	논리 OR 연산. 둘 중 하나만 참이여도 참
not	논리 NOT 연산. 논리 상태를 반전

Ex)

```
3<6 and 2<3
```

```
True
```

```
0>1 or 9>7
```

```
True
```

```
not False
```

```
True
```

➡ 비교연산자를 사용, 복수의 값들의 비교가 참인지 거짓인지에 대한 값 반환

➡ 비교되는 복수의 값들에 대한 설정이 논리 연산

D&A

Basic Session 2차시

03 - 문자열 포매팅

2022 / 03 / 29

D&A 운영진 이경욱

03. 문자열 포매팅

문자열 포매팅이란?

- 문자열 안에 어떠한 값을 삽입하는 방법
- 문자열 안의 특정한 값을 바꿔야 할 경우

Ex)

```
print("저는 3학년입니다.")
```

```
저는 3학년입니다.
```

➡ 이 때 학년만 다르게 사용한다면??

```
print("저는 %s학년입니다."%3)
print("저는 %s학년입니다."%2)
```

```
저는 3학년입니다.
저는 2학년입니다.
```

➡ 포매팅 기법으로 숫자만 변경

1. 포맷 코드를 사용한 문자열 포매팅

%s	문자열(String)
%c	문자 1개(character)
%d	정수(Integer)
%f	부동소수(floating-point)
%o	8진수
%x	16진수
%%	Literal % (문자 % 자체)

➡ 입력하려는 값에 해당하는 자료형을 포맷코드로 식별

Ex)

```
print("제 이름은 %s이고 학년은 %d학년입니다."%("이경욱", 3))
```

```
제 이름은 이경욱이고 학년은 3학년입니다.
```

➡ 여러 값은 콤마로 구별하고 ()로 둘러싸기

03. 문자열 포매팅

1. 포맷 코드를 사용한 문자열 포매팅
- 유용하게 사용하기

1) 정렬과 공백

```
"%10s" %"hi"
'        hi'
```

➡ 전체 길이 10인 공간에 hi를 오른쪽 정렬로 대입

```
"%-10s경욱" %"hi"
'hi        경욱'
```

➡ 왼쪽 정렬로 경욱, hi를 포함한 문자열길이 10

2) 소수점 표현

```
"%0.4f"%12.584524
'12.5845'
```

➡ 소수점 넷째자리까지 반환을 의미

3) %대입하기

```
"%0.2f%%"%1.123
'1.12%'
```

➡ %를 넣으려면 "%%"대입

2. format 함수를 사용한 문자열 포매팅

```
"제 이름은 {0}이고 학년은 {1}학년입니다.".format("이경욱", 3)
'제 이름은 이경욱이고 학년은 3학년입니다.'
```

➡ 값을 삽입할 부분에 {인덱스 값}, 문자열.format() 괄호 안에 삽입할 값 포함

➡ 인덱스 값은 format()안의 값들에 대한 인덱스 값임

2. format 함수를 사용한 문자열 포매팅 – 다양한 사용법

```
name = "손흥민"
birth = 1992
"저는 {0}이고 {1}년생입니다.".format(name, birth)
'저는 손흥민이고 1992년생입니다.'
```

➡ 변수로 삽입

```
"저는 {name}이고 {birth}년생입니다.".format(name = "손흥민", birth = 1992)
'저는 손흥민이고 1992년생입니다.'
```

➡ 인덱스가 아닌 이름으로 넣기

03. 문자열 포매팅

3. f 문자열 포매팅(f-string 문법)

```
name = "손흥민"
birth = 1992

print(f"저는 {name}이고 {birth}년생입니다.")
```

```
저는 손흥민이고 1992년생입니다.
```

➡ 문장 앞에 f접두사 붙임

➡ {} 내에 변수 또는 값 직접 삽입

➡ {} 내에서 연산 가능

3. f 문자열 포매팅으로 dictionary value 참조하기

```
user = {"이름" : "백종원", "나이" : 28, "직업" : "개 훈련사"}
f"제 이름은 {user['이름']}이고, 나이는 {user['나이']}입니다."
```

```
'제 이름은 백종원이고, 나이는 28입니다.'
```

➡ Dictionary[“key”]를 해당 위치에 삽입해 참조

04 – 메소드

04. 메소드

파이썬의 함수란?

```
print("안녕하세요")
```

```
안녕하세요
```

```
a = abs(-12)
a
```

```
12
```

➡ 함수에 정의된 기능을 구현

➡ 함수의 결과를 변수에 대입 가능

➡ 함수명으로 독립적으로 호출 가능

➡ function_name(input)형태로 사용

메소드란?

```
a = [4, 3, 2, 3, 1]
a.sort()
a
```

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

➡ 객체(특정 자료형을 갖는 값 또는 그 변수)와 연관되어 사용하는 함수

➡ Object.method_name(input or input없음)로 사용

함수가 메소드를 포함하는 포괄적인 개념임!!

04. 메소드 - 문자열 메소드

문자열 메소드 예시

```
a = "python is not hard"
a.count("n")  # 문자열 내 input의 개수세기
```

```
2
```

```
a.find("n")   # 문자열 내 input의 첫번째 요소의 인덱스값
```

```
5
```

```
a.replace("python", "math") # 문자열의 첫번째input을 두번째 input으로 대체
```

```
'math is not hard'
```

```
a.split(" ")          # 문자열을 input기준으로 나누어 리스트에 저장
```

```
['python', 'is', 'not', 'hard']
```

```
",".join("abcd")      # input으로 들어간 문자열 사이에 ""내의 값 삽입
```

```
'a,b,c,d'
```

대표적인 문자열 메소드

	문자열 메소드	메소드 설명
0	a.count(x)	문자열 내 input(x)의 개수 세기
1	a.find(x)	문자열 내 input(x)과 등일한 첫번째 요소의 인덱스값
2	a.replace('x', 'y')	문자열 내 첫 번째 input값(x)을 두 번째 input값(y)으로 대체
3	a.split()	문자열을 input 기준으로 나누어 리스트에 저장
4	'x'.join(a)	input으로 들어간 문자열(a)의 각 요소 사이에 "(x) 삽입
5	a.lower()	문자열의 대문자를 소문자로 바꿈
6	a.upper()	문자열의 소문자를 대문자로 바꿈
7	a.swapcase()	대문자는 소문자로, 소문자는 대문자로 바꿈
8	a.capitalize()	문자열의 첫 글자를 대문자로, 나머지는 소문자로 바꿈
9	a.lstrip()	문자열의 왼쪽 공백을 없앰
10	a.rstrip()	문자열의 오른쪽 공백을 없앰
11	a.strip()	문자열의 양쪽 공백을 없앰
12	len(a)	문자열의 길이를 숫자형으로 반환
13	a.startswith(x)	문자열이 input값(x)로 시작하는지 여부를 bool형태로 반환
14	a.endswith(x)	문자열이 input값(x)로 끝나는지 여부를 bool형태로 반환
15	a.isalnum()	문자열이 숫자와 알파벳으로 이루어져있으면 True, 아니면 False
16	a.isalpha()	문자열이 문자(영어, 한글)만 포함하고 있으면 True, 아니면 False
17	a.isdigit()	문자열이 숫자로만 이루어져있으면 True, 아니면 False

04. 메소드 - 리스트 메소드

리스트 메소드 예시

```
a = [2, 1, 3]
a.append(4)    #리스트에 요소 추가
a
```

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

```
a.reverse()    #리스트를 요소값 기준이 아닌
               # 현재 순서에서 뒤집기
a
```

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

```
a.sort()
a              #리스트 요소값으로 정렬
```

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

```
a.index(2) # input의 리스트 내 인덱스값 반환
```

```
1
```

```
a.insert(0, 6) # 첫번째 input 인덱스 자리에
               # 두번째 값 삽입
a
```

```
[6, 1, 2, 3, 4]
```

```
a.remove(6)  # input에 해당하는 요소 삭제
a
```

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

```
a.count(1)  # input에 해당하는 요소 개수 세기
```

```
1
```

```
a.extend([5, 6, 7])
a          # input으로 리스트를 받아 리스트를 확장
```

```
[1, 2, 3, 4, 5, 6, 7]
```

대표적인 리스트 메소드

	리스트 메소드	메소드 설명
0	a.append(x)	리스트에 input값(x)추가
1	a.reverse()	리스트를 현재 순서에서 뒤집기
2	a.sort()	리스트 요소값 기준으로 정렬
3	a.index(x)	input값(x)의 인덱스 값 반환
4	a.insert(x, y)	x에 해당하는 인덱스값 자리에 y 삽입
5	a.remove(x)	input값(x)에 해당하는 리스트 내 요소 삭제
6	a.count(x)	input값(x)에 해당하는 요소 개수 세기
7	a.extend(x)	리스트에 input(x)으로 들어온 리스트 확장
8	del(a[0])	리스트의 해당 인덱스값의 요소 삭제
9	a.pop()	리스트의 마지막 값 출력 후
10	len(a)	리스트의 요소 개수 반환

04. 메소드 - 딕셔너리 메소드

딕셔너리 메소드 예시

```
user = {"이름" : "손흥민", "나이" : "29", "직업" : "축구선수"}
user.keys()
```

```
dict_keys(['이름', '나이', '직업'])
```

```
user.values()
```

```
dict_values(['손흥민', '29', '축구선수'])
```

```
user.items()
```

```
dict_items([('이름', '손흥민'), ('나이', '29'), ('직업', '축구선수')])
```

```
user.get("이름")  # user["이름"]과 동일
```

```
'손흥민'
```

➡ dict_keys, dict_values같은 경우 고유한 자료형이므로 List로 쓰고싶을 때는 다음과 같이 변경해서 사용

```
type(user.items())
```

```
dict_items
```

```
type(list(user.items()))
```

```
list
```

대표적인 딕셔너리 메소드

	딕셔너리 메소드	메소드 설명
0	a.keys()	딕셔너리의 모든 key 출력
1	a.values()	딕셔너리의 모든 value 출력
2	a.items()	딕셔너리의 모든 key:value 조합을 튜플형태로 반환
3	a[key]	딕셔너리의 key에 해당하는 value 출력
4	a.get(key)	딕셔너리의 key에 해당하는 value 출력
5	del a[key]	딕셔너리의 key와 대응되는 value 쌍 제거

04. 메소드 - 자주 쓰이는 기본적인 함수

파이썬의 대표적인 내장함수

- enumerate()

```
a = ["이것은", "파이썬", "이다"]
for i, j in enumerate(a):
    print(i, j)
```

```
0 이것은
1 파이썬
2 이다
```

➡ 순서가 있는 자료형(리스트, 문자열 등)을 넣으면 요소와 인덱스값을 매칭시켜주는 함수

- range()

```
list(range(0, 5))
```

```
[0, 1, 2, 3, 4]
```

➡ 입력받은 값의 범위 내의 값들을 반환

➡ 시작 범위를 지정하지 않을 시 0으로 자동설정, 범위의 끝은 포함X

- zip()

```
list(zip([1, 2, 3], [4, 5, 6]))
```

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

➡ 동일한 길이의 자료형들의 요소를 같은 인덱스값끼리 매칭시켜주는 함수

```
print("안녕하세요") # input값을 출력
```

```
안녕하세요
```

```
len([1, 2, 3])  # input에 해당하는 자료형의 길이를 반환
```

```
3
```

```
abs(-12)   # input으로 들어온 숫자형의 절대값을 반환
```

```
12
```

```
round(3.14159, 2)  # 첫 번째 input값으로 들어온 실수를
                   #  두 번째 input값 자리까지 반올림
```

```
3.14
```

```
str(1234)  # input을 문자열로 바꿔줌
```

```
'1234'
```

```
int("21")   # input을 정수형으로 바꿔줌
```

```
21
```

```
float(21)  # input을 실수형으로 바꿔줌
```

```
21.0
```

```
list("123456") # 반복가능한 input을 요소별로 나누어 list로 반환
```

```
['1', '2', '3', '4', '5', '6']
```

```
sum([1, 2, 3, 5]) # 입력받은 리스트나 튜플의 요소 합을 반환
```

```
11
```

첨부자료 출처

폰트

네이버 글꼴 모음 _ 나눔 스퀘어 사용
출처 : https://hangeul.naver.com/font

D&A

Basic Session 2차시 파이썬기초1

Thank You.

2022 / 03 / 15

D&A 운영진 이경욱