[SK] 데이터 분석을 위한 Python 기초

01 개요
02 자료구조
03 집합형 자료구조
04 문자열
05 연산
06 제어
07 함수
08 패키지와 모듈

01. 개요

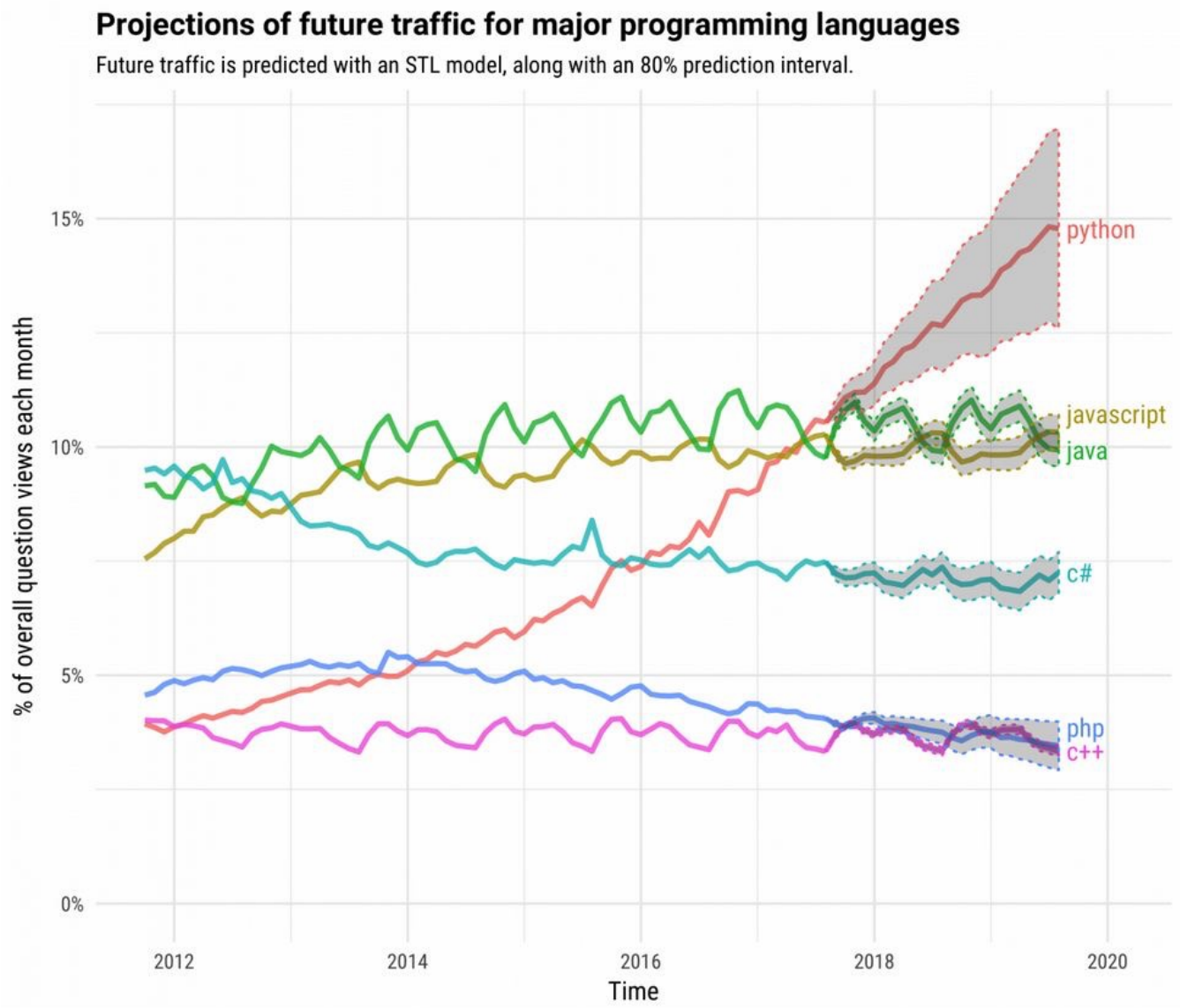
파이썬의 탄생

- 생년월일: 1991년
- 인터프리터식 언어
- 파이썬 2.x 버전은 지고, 파이썬 3.6 이상 버전을 주로 사용
- 창시자: 귀도 반 로섬 (Guido van Rossum)
- 구글, 드롭박스 근무, 현재 은퇴

01. 개요

Why Python?

- 빠르게 성장하는 프로그래밍 언어!



01. 개요

왜 파이썬은 인기있는 언어가 되었을까?

- 무료 오픈소스
- 쉽고 간결하다
- 풍부한 라이브러리로 생산성이 매우 높다
  - 웹
  - 앱
  - 차트
  - 통계 분석
  - 데이터 분석
  - 머신러닝
  - 딥러닝

01. 개요

가성비의 파이썬

하나의 언어를 배워서
데이터분석 뿐만 아니라,
다방면으로 활용할 수 있는 언어

“파이썬”

01 개요
02 자료구조
03 집합형 자료구조
04 문자열
05 연산
06 제어
07 함수
08 패키지와 모듈

02. 자료구조

print (기본 입출력)

- print(value, sep=‘ ’, end=‘₩n’, …)
  - value을 출력
  - sep : 출력되는 value들 사이 분리 기호
  - end : 출력 문자열 마지막 문자
    - 기본 값: 줄바꿈 (₩n)

```
print('오늘 파이썬 배우기 좋은 날이네!')
```

오늘 파이썬 배우기 좋은 날이네!

```
print(''오늘은'', ''일요일'')
```

오늘은 일요일

```
print(''오늘은'', ''일요일'', end='+')
```

오늘은 일요일+

02. 자료구조

변수

- 변수란 자료형의 값을 저장하는 공간
- 변수명 = 변수에 저장할 값
- 변수의 타입 확인

  type(b)

02. 자료구조

변수

- 명명 규칙

① 첫 번째 글자 : 영문자, _(under score)

② 두 번째 글자 이후 : 영문자, 숫자, _(under score)

③ 글자 수 길이 무제한

④ Unicode (한글가능)

⑤ Keyword 제외(변수 이름으로 사용 불가능)

False, elif, lambda, None, else, True, except, not, and, finally, or, as, for

pass, assert, from, raise, break, global, return, class, if, try, continue, import, while,

def, in, with, del, is, yield

02. 자료구조

변수

- 허용되지 **않는** 경우
    1. 숫자가 먼저 오는 경우 (예. 1a)
    2. _(under score)를 제외한 특수문자 (예. a$)
    3. 변수명 사이에 공백 (예. a b)

02. 자료구조

데이터 타입

분류	타입	타입명	리터럴 예시
숫자	정수	int	1, 4, 10
	실수	float	3.156
	~~복소수~~	~~complex~~	~~7+7j~~
참, 거짓	부울 (불리언)	bool	True, False
열거형(Sequence)	문자열	Str	“Hello World”
집합	리스트	list	[10, 20, 30, 40]
	튜플	tuple	(1, 2, 3, 4)
	세트	set	{3, 5, 9, 10}
사전	사전 (딕셔너리)	dict	{‘a’:1, ‘b’:2}

02. 자료구조

숫자형 타입 확인과 변경

- type 함수를 사용하여 데이터 타입 검사 가능

```
type(1)
```

```
int
```

```
type(3.14)
```

```
float
```

```
type(''안녕'')
```

```
str
```

```
type(True)
```

```
bool
```

02. 자료구조

숫자형 타입 확인과 변경

- type 변경

```
a = 1
```

```
type(a)
```

```
int
```

```
float(a)
```

```
1.0
```

```
b = 3.99
```

```
int(b)
```

```
3
```

```
c = True
```

```
int(c)
```

```
1
```

```
d = False
```

```
int(d)
```

```
0
```

```
float(True)
```

```
1.0
```

01 개요
02 자료구조
03 집합형 자료구조
04 문자열
05 연산
06 제어
07 함수
08 패키지와 모듈

03. 집합형 자료구조

데이터 타입

분류	타입	타입명	리터럴 예시
숫자	정수	int	1, 4, 10
	실수	float	3.156
	~~복소수~~	~~complex~~	~~7+7j~~
참, 거짓	부울 (불리언)	bool	True, False
열거형(Sequence)	문자열	str	“Hello World”
집합	리스트	list	[10, 20, 30, 40]
	튜플	tuple	(1, 2, 3, 4)
	세트	set	{3, 5, 9, 10}
사전	사전 (딕셔너리)	dict	{‘a’:1, ‘b’:2}

03. 집합형 자료구조

리스트, 집합, 세트, 사전

- 리스트(list)
  - 여러 가지 형태의 변수를 하나의 순서가 있는 배열로 묶어놓은 형태
- 튜플(tuple)
  - 조회만 가능하고 변경이 불가능한 리스트
- 세트(set)
  - 순서가 없고 중복을 허용하지 않음
- 사전(dictionary)
  - Key와 value 형태로 데이터를 저장함

03. 집합형 자료구조

리스트(list)

- 리스트(list) 규칙
  - list는 다양한 type의 데이터를 집합으로 가집니다
  - list안에 list도 허용합니다.
  - list는 순서(order)의 개념이 존재합니다.

```
a = []
```

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

```
a = [1, 'hello', 3, 3.14, True]
a
```

```
a = [1, 'hello', 3, 3.14, [6, 7, '8']]
a
```

03. 집합형 자료구조

리스트(list)

- 리스트 생성
  - mylist=[]
  - mylist=[1, 2, 3, 4, 5]
  - mylist=list()

03. 집합형 자료구조

리스트(list)

- 리스트는 다양한 메서드(method) 혹은 함수를 지원하며 메서드를 활용하여 요소를 추가, 삭제 및 변경
  - 메서드(method): 객체(object)가 포함하는 함수 혹은 기능입니다. (함수에 대한 내용은 추후 다룹니다)
  - . 점 연산자로 함수를 실행할 수 있습니다.
- append(): 값 추가
  - append(값) : 리스트의 맨 마지막에 '값' 추가

```
mylist.append(7)
mylist.append(7)
mylist.append(7)
mylist.append(3)
mylist.append(5)
mylist.append(2)
mylist
```

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

03. 집합형 자료구조

리스트(list)

- insert(): 값 추가
  - Insert(index, 값) : 위치(index)에 '값' 삽입

```
mylist.insert(1, 100)
mylist
```

```
[1, 100, 100, 100, 7, 7, 7, 3, 5, 2]
```

03. 집합형 자료구조

리스트(list)

- 삭제
  - remove(값): 첫 번째 나오는 '값' 삭제

```
mylist.remove(7)
mylist
```

```
[100, 100, 100, 7, 7, 5, 3, 2, 1]
```

  - pop(index) : index 번째 요소를 반환 및 삭제

```
mylist.pop(1)
mylist
```

```
100
```

03. 집합형 자료구조

리스트(list)

- len(): 리스트 크기
  - len(리스트)

```
myList = [10, 20, 30, 40, 50]
len(myList)
```

```
5
```

03. 집합형 자료구조

리스트(list)

- 정렬
  - sort( ) : 정렬

```
mylist.sort()
mylist
```

```
[1, 2, 3, 5, 7, 7, 7, 100, 100, 100]
```

  - sort(reverse=True) : 역정렬(내림차순 정렬)

```
mylist.sort(reverse=True)
mylist
```

```
[100, 100, 100, 7, 7, 7, 5, 3, 2, 1]
```

03. 집합형 자료구조

리스트(list)

- extend() : 리스트 확장

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

```
a.extend([4, 5])
```

```
5
```

```
a
```

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

03. 집합형 자료구조

리스트(list)

- 색인: index
  - mylist=[1, 2, 3, 4]

```
mylist([0])
```

1

```
mylist([3])
```

4

```
mylist([4])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-3-88b11041aa4f> in <module>()
----> 1 mylist[4]

IndexError: list index out of range
```

```
mylist([-1])
```

4

```
mylist([-3])
```

2

```
mylist([-5])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-6-7f152f66d2d3> in <module>()
----> 1 mylist[-5]

IndexError: list index out of range
```

03. 집합형 자료구조

리스트(list)

- 리스트 색인하여 값 수정

```
mylist
```

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

```
mylist[0] = 100
mylist
```

```
mylist[-1] = 300
mylist
```

03. 집합형 자료구조

리스트(list)

- 중첩 리스트 색인

```
mylist = [[1,2,3], [4,5,6], [7,8,9]]
mylist
```

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

```
mylist[1]
```

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

```
mylist[1][1]
```

```
5
```

03. 집합형 자료구조

리스트(list)

- 슬라이싱 (추출)
  - mylist=[100, 200, 300, 400, 500]

```
mylist = [100, 200, 300, 400, 500]
```

```
mylist([1:3])
```

```
[200, 300]
```

```
mylist([-3: -1])
```

```
[300, 400]
```

```
mylist([2:])
```

```
[300, 400, 500]
```

```
mylist([:3])
```

```
[100, 200, 300]
```

```
mylist([-3:])
```

```
[300, 400, 500]
```

03. 집합형 자료구조

리스트(list)

- 슬라이싱 (추출)
  - mylist[시작인덱스 : 끝 인덱스 : 증가치]

```
mylist
```

```
[100, 200, 300, 400, 500]
```

```
mylist [::-1]
```

```
[500, 400, 300, 200, 100]
```

```
mylist [1::2]
```

```
[200, 400]
```

```
mylist [::2]
```

```
[100, 300, 500]
```

```
mylist [3:1:-1]
```

```
[400, 300]
```

03. 집합형 자료구조

튜플(tuple)

- 데이터 변경이 불가능한 리스트. 즉, 요소에 대한 **수정, 삭제, 변경**이 **불가**
- 리스트는 대괄호[ ]로 둘러싸여 있지만 튜플은 소괄호( )로 둘러싸임
- 튜플(tuple)의 생성
  - mytuple = (1, 2, 3)
  - mytuple = 1, 2, 3

03. 집합형 자료구조

튜플(tuple)

- 리스트는 값의 추가, 수정, 삭제가 가능하지만 튜플은 불가능함
- 튜플의 인덱싱과 슬라이싱은 리스트와 동일
- 값들의 수정이 불필요할 경우 리스트보다 튜플을 사용하는 것이 더 효율적

03. 집합형 자료구조

세트(set)

- 집합은 중괄호{ } 안에 쉼표(,)로 값들을 구분
- **중복을 허용하지 않으며** 순서가 없음

```
myset = set()
```

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

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

```
myset = {1, 1, 1, 2, 2, 2, 3, 4, 5}
myset
```

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

03. 집합형 자료구조

세트(set)

- add(값) : 값 추가

```
myset = set()
```

```
myset.add(1)
myset.add(2)
myset.add(3)

myset.add(1)
myset.add(2)
myset.add(3)

myset.add(1)
myset.add(2)
myset.add(3)
```

```
myset
```

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

03. 집합형 자료구조

세트(set)

- update(집합): 여러 개 값 일괄 추가

```
myset
```

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

```
myset.update([4, 5, 6])
```

```
myset
```

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

03. 집합형 자료구조

세트(set)

- 값 제거: remove(값)

```
myset
```

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

```
myset.remove(2)
```

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

03. 집합형 자료구조

세트(set)

- 세트(set)는 집합의 특성을 가짐
  - 교집합(intersection), 합집합(union), 차집합(difference)
- 교집합 (intersection)

```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7}
```

```
a & b
```

```
{3, 4, 5}
```

```
a.intersection(b)
```

```
{3, 4, 5}
```

03. 집합형 자료구조

세트(set)

- 합집합 (union)

```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7}
```

```
a | b
```

```
{1, 2, 3, 4, 5, 6, 7}
```

```
a.union(b)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

03. 집합형 자료구조

세트(set)

- 차집합 (difference)

```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7}
```

```
a - b
```

```
{1, 2}
```

```
b - a
```

```
{6, 7}
```

```
a.difference(b)
```

```
{1, 2}
```

03. 집합형 자료구조

딕셔너리(dictionary)

- 딕셔너리는 key와 value의 쌍 여러 개와 { }로 구성되어 있음

  mydict = {'a': 1, 'b': 2, 'c': 3, 'd':4}

- key값을 통해 value를 얻음
- 순서가 없음
- key값은 **고유한 값**이어야 함
- type은 dictionary가 아닌 축약형 **dict**로 출력

```
type(mydict)
```

```
dict
```

03. 집합형 자료구조

딕셔너리(dictionary)

- 딕셔너리 값 조회 (key를 이용하여 value 조회)

```
mydict = {'a': 1, 'b': 2, 'c': 3}
```

```
mydict['b']
```

2

```
mydict['d']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/tmp/ipykernel_3246769/908337734.py in <module>
----> 1 mydict['d']

KeyError: 'd'
```

03. 집합형 자료구조

딕셔너리(dictionary)

- 모든 key 조회: keys()

```
mydict.keys()
```

```
dict_keys([‘a’, ‘b’, ‘c’])
```

```
mydict = {'a': 1, ‘b’: 2, 'c’: 3}
mydict
```

```
{‘a’: 1, ‘b’: 2, ‘c’: 3}
```

- 모든 값 조회: values()

```
mydict.values()
```

```
dict_values([1, 2, 3])
```

- 모든 key, value 모두 조회: items()

```
mydict.items()
```

```
dict_items([(‘a’, 1), (‘b’, 2), (‘c’, 3)])
```

03. 집합형 자료구조

딕셔너리(dictionary)

- 값 추가

```
mydict = dict()
```

```
mydict['apple'] = 123
```

```
mydict[0] = 2
```

```
mydict
```

```
{'apple': 123, 0: 2}
```

03. 집합형 자료구조

딕셔너리(dictionary)

- update(딕셔너리): 다중 업데이트

```
mydict
```

```
{'파인애플': 1500, '망고': 3500, '배': 1000}
```

```
fruit = {
    '사과' : 2000
    '딸기' : 3000
    '수박' : 5000
}
```

```
mydict.update(fruit)
```

```
mydict
```

```
{'파인애플': 1500, '망고': 3500, '배': 1000, '사과': 2000, '딸기': 3000, '수박': 5000}
```

03. 집합형 자료구조

딕셔너리(dictionary)

- 값 변경: 키(key)에 새로운 값(value) 대입

```
mydict = {'a': 100, 'b': 200, 'c': 300, 'd': 400, 'e': 500}
mydict
```

```
{'a': 100, 'b': 200, 'c': 300, 'd': 400, 'e': 500}
```

```
mydict['a'] = 900
```

```
mydict
```

```
{'a': 900, 'b': 200, 'c': 300, 'd': 400, 'e': 500}
```

03. 집합형 자료구조

딕셔너리(dictionary)

- pop(): 요소 제거

```
mydict
```

```
{‘키위’: 1500,‘사과’: 2000, ‘수박’:5000}
```

```
mydict.pop('사과')
```

```
2000
```

```
mydict
```

```
{‘키위’: 1500, ‘수박’:5000}
```

01 개요
02 자료구조
03 집합형 자료구조
04 문자열
05 연산
06 제어
07 함수
08 패키지와 모듈

04. 문자열

문자열(str)

- 데이터 분석에 있어서 문자열(텍스트) 데이터의 중요성
- 자연어처리 데이터는 대부분이 텍스트 데이터로 이루어져 있습니다.
- 우리가 흔히 접하는 Excel 혹은 table 데이터 안에도 수많은 텍스트 데이터가 존재합니다.
- 게다가 우리나라는 영어와 더불어 한글까지 추가로 처리 할 수 있어야 합니다.

특성

- 문자열 역시 리스트(list), 튜플(tuple)과 마찬가지로 순차적인(sequence)형 자료구조를 가집니다.
- 문자열은 불변(immutable) 객체입니다.

04. 문자열

문자열(str)

- 문자열 생성
  - 큰 따옴표(“ ”)

```
''안녕하세요? 반갑습니다.''
```

‘ 안녕하세요? 반갑습니다. ’

  - 작은 따옴표(‘ ’)

```
'안녕하세요? 반갑습니다.'
```

‘ 안녕하세요? 반갑습니다. ’

04. 문자열

문자열 생성

- 문자열 만들기
  - 큰 따옴표를 3개 연속 (“”” ”””) / 작은 따옴표 3개 연속(‘’’ ’’’)
    - 여러 줄 문자열 표현할 때

```
"""안녕하세요? 반가워요
내이름은 파이썬 입니다.

웰컴!
"""
```

안녕하세요? 반가워요
내이름은 파이썬입니다.

웰컴!

```
'''안녕하세요? 반가워요
내이름은 파이썬 입니다.'''
```

안녕하세요? 반가워요
내이름은 파이썬입니다.

04. 문자열

문자열 출력

- print 출력

```
print('헬로우 파이썬')
```

```
헬로우 파이썬
```

```
print('문자열 첫째', '그리고, 둘째')
```

```
문자열 첫째 그리고, 둘째
```

04. 문자열

문자열 포맷팅

- %

```
'안녕하세요? %s' % ('반갑습니다.')
```

‘ 안녕하세요? 반갑습니다. ‘

```
'안녕하세요? %.3f' % (0.123456)
```

‘ 안녕하세요? 0.123 ’

```
'안녕하세요? %d' % (12345)
```

‘ 안녕하세요? 12345 ’

```
'안녕하세요? %c' % ('a')
```

‘ 안녕하세요? a ’

04. 문자열

문자열 포맷팅

- {} 와 .format()

```
'웰컴투? {}'.format('파이썬.')
```

‘ 웰컴투? 파이썬. ’

```
'비밀번호 {}'.format(486)
```

‘ 비밀번호 486 ’

```
' 원주율? {:.2f} ' .format(3.141592)
```

‘ 원주율? 3.14 ’

04. 문자열

문자열 포맷팅

- f 문자열 포맷팅 (python 3.6 이상 지원)

```
name = '펭수'
age = 10
print(f'나의 이름은 {name}입니다. 나이는 {age} 살입니다.')
```

나의 이름은 펭수입니다. 나이는 10 살입니다.

```
print(f'내년에 저는 {age+1} 살입니다.')
```

내년에 저는 11살입니다.

```
d = {'name': '펭수', 'age': 10}
```

```
print(f"반가워요. 저는 {d['name']}입니다. 저의 나이는 {d['age']} 살입니다.")
```

반가워요. 저는 펭수입니다. 저의 나이는 10살입니다.

04. 문자열

문자열(str)

- 길이

```
len('banana')
```

6

```
len('banana pen')
```

10

```
len('한글')
```

2

```
len('한글 킹왕짱')
```

6

04. 문자열

문자열(str)

- 색인(indexing) & 슬라이싱(slicing)

```
a = 'Python is my life'
```

```
a[0]
```

‘P’

```
a[:6]
```

‘Python’

```
a[3:6]
```

‘hon’

```
a[-2]
```

‘f’

```
a[:-3]
```

‘Python is my l’

```
a[-4:]
```

‘life’

04. 문자열

문자열(str)

- 수정 불가

```
a = 'Python is my life'
```

```
a[2] = 'Y'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-f2db0e8f1888> in <module>()
----> 1 a[2] = 'Y'

TypeError: 'str' object does not support item assignment
```

04. 문자열

문자열(str)

- 결합과 반복

```
a = '반갑습니다'
b = '웰컴 투 파이썬'
```

```
print(a + b)
```

반갑습니다! 웰컴 투 파이썬

```
print(a + '???')
```

반갑습니다!???

```
print(a * 2)
```

반갑습니다! 반갑습니다!

```
print('===' * 10)
```

==========

```
print(a + b)
print('===' * 10)
print(b + ', ' + a)
```

반갑습니다! 웰컴 투 파이썬

==========

웰컴 투 파이썬, 반갑습니다!

04. 문자열

문자열(str)

- 분리: split()

```
a = 'This is a pen'
```

```
a.spilt( ' ' )
```

['This', 'is', 'a', 'pen']

```
a.spilt()
```

['This', 'is', 'a', 'pen']

```
a = 'This-is-a-pen'
```

```
a.spilt( '-' )
```

['This', 'is', 'a', 'pen']

04. 문자열

문자열(str)

- 합치기: join()

```
'-'.join(['010', '1234', '5678'])
```

['010-1234-5678']

```
'-'.join('ABCDE')
```

['A-B-C-D-E']

04. 문자열

문자열(str)

- 대문자: upper() / 소문자: lower()

```
a = 'My name is Teddy'
```

```
a.lower()
```

‘my name is teddy’

```
a.upper()
```

‘MY NAME IS TEDDY’

```
a = '한글엔 대소문자가 없어요ㅠ'
```

```
a.lower()
```

‘한글엔 대소문자가 없어요ㅠ’

```
a.upper()
```

‘한글엔 대소문자가 없어요ㅠ’

04. 문자열

문자열(str)

- 바꾸기: replace()

```
a = '01-sample.png'
```

```
a.replace('.png', '.jpg')
```

‘01-sample.jpg’

04. 문자열

문자열(str)

- 공백제거: strip()

```
a = '    01-sample.png          '
```

```
a.strip()
```

```
'01-sample.png'
```

01 개요
02 자료구조
03 집합형 자료구조
04 문자열
05 연산
06 제어
07 함수
08 패키지와 모듈

05. 연산자

사칙연산 + 그 밖의 연산

연산자	결과
x + y	x 더하기 y
x - y	x 빼기 y
x * y	x 곱하기 y
x / y	x 나누기 y

05. 연산자

그 밖의 연산

- // (floor division): 몫
- % (modulus): 나머지
- ** 제곱 연산

연산자	결과
x // y	x를 y로 나눈 몫
x % y	X를 y로 나눈 나머지
x ** y	x의 y승

05. 연산자

그 밖의 연산

- // (floor division)

```
a = 10
b = 3
```

```
a / b
```

```
3.33333333333333
```

```
a // b
```

```
3
```

```
print(11 / 3)
print(11 // 3)
```

```
3.66666666666666
3
```

05. 연산자

그 밖의 연산

- % (modulus)

```
a = 10
b = 3
```

```
a % b
```

1

05. 연산자

그 밖의 연산

- ** 제곱 연산

```
a = 10
b = 3
```

```
a ** b
```

```
1000
```

```
2 ** 10
```

```
1024
```

05. 연산자

사칙연산

- 괄호 먼저 연산

```
10 + 2 * 5
```

```
20
```

```
(10 + 2) * 5
```

```
60
```

05. 연산자

비교연산자

비교 연산자	뜻
<	작은
<=	작거나 같은
>	큰
>=	크거나 같은
==	같은
!=	같지 않은

05. 연산자

논리연산자

논리 연산자	결과
x or y	x, y 중 하나만 참이면 참
x and y	x, y 둘 다 참이어야 참
not x	x가 참이면 거짓, x가 거짓이면 참

01 개요
02 자료구조
03 집합형 자료구조
04 문자열
05 연산
06 제어
07 함수
08 패키지와 모듈

06. 제어

If문

- 조건문이란?

  명시한 자료형 조건 (참/거짓) 이 참인지 거짓인지에 따라
  달라지는 계산이나 상황(로직, logic)을 수행하는 것

06. 제어

if, else

- 조건에 따라 분기

```
if person ==  '남자' :
    print( '당신은 남자입니다.' )
else:
    print( '당신은 여자입니다.' )
```

  - if 문을 만들 때는 **if 조건문** 아래부터 쓸 것
  - if 에 속하는 모든 문장에 대해 들여쓰기
  - 조건문 끝에는 : 이 와야함
  - **if** 명제가 **True(참)** 일 때, 로직을 수행
  - **else** 는 if 명제가 **False(거짓)**일 경우 수행

06. 제어

조건문

- if, elif, else

```
if 3 > 5:
    print('if 구문')
elif 3 > 4:
    print('elif 1 구문')
elif 3 > 5:
    print('elif 2 구문')
elif 3 < 6:
    print('elif 3 구문')
else:
    print('이것도 저것도 아니다')
```

```
elif 3 구문
```

- if 문이 거짓인 경우 다음 elif 조건문 실행
- 다음 elif 조건문도 거짓인 경우 다음 elif 문 실행
- 모든 elif 조건문이 거짓인 경우 else 구문 실행

06. 제어

조건문

- 모두 참인 조건인 경우?

```
if 3 < 5:
    print('if 구문')
elif 3 < 4:
    print('elif 1 구문')
elif 3 < 5:
    print('elif 2 구문')
elif 3 < 6:
    print('elif 3 구문')
else:
    print('이것도 저것도 아니다')
```

if 구문

- 가장 첫 번째 '참' 인 구문을 실행

06. 제어

삼항연산자

- (참인 값) if 조건 else (거짓인경우 값)

```
age = 35

"30세 이상입니다." if age >= 30 else "30세 이하입니다."
```

```
'30세 이상입니다.'
```

```
age = 20

"30세 이상입니다." if age >= 30 else "30세 이하입니다."
```

```
'30세 이하입니다.'
```

06. 제어

반복문

- 반복문이란?

  반복문이란 프로그램 내에서 똑같은 명령을 일정 횟수만큼 반복하여 수행하도록 제어하는 명령문입니다.

  프로그램이 처리하는 대부분의 코드는 반복적인 형태가 많으므로, 가장 많이 사용되는 제어문 중 하나입니다.

  list, dict, set 등 집합에 대한 순회를 돌며 일을 처리할 때 많이 쓰입니다.

- 파이썬에는 for ~ in 구문 과 while 구문이 대표적인 반복문 입니다.

06. 제어

for ~ in

- 기본구조

  for <변수> in <iterable(list, tuple, str, …)>:

  (들여쓰기) 로직

```
fruits = [ ‘ apple ’ ,  ‘ banana ’ ,  ‘ kiwi ’ ]
for f in fruits:
    print(f)
```

apple

banana

kiwi

06. 제어

for ~ in

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

```
print(mylist[0])
print(mylist[1])
print(mylist[2])
print( ' ... ' )
print(mylist[8])
print(mylist[9])
```

```
1
2
3
…
9
10
```

```
for i in mylist:
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

```
for i in (1, 2, 3, 4, 5):
    print(i)
```

```
1
2
3
4
5
```

06. 제어

dictionary 반복문

```
mydict = {'헐크': 50, '아이언맨': 60 , '펭수': 70}
```

```
for key in mydict.keys():
    print(key)
```

```
헐크
아이언맨
펭수
```

```
for value in mydict.values():
    print(value)
```

```
50
60
70
```

```
for key, value in mydict.items():
    print(key, value)
```

```
헐크 50
아이언맨 60
펭수 70
```

06. 제어

range()

- 특정 범위의 정수를 표현할 때 사용
- 수정이 불가
- 생성방법
  - range(start, stop, step)

```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
for i in range(2, 9):
    print(i)
```

```
2
3
4
5
6
7
8
```

```
for i in range(1, 10, 2):
    print(i)
```

```
1
3
5
7
9
```

06. 제어

for ~ in 구문을 중첩하여 사용할 경우

- for ~ in 문의 중첩

```
for i in range(1, 4):
    for j in range(1, 4):
        print(f'(i={i})+ (j={j})= {i * j}')
    print('===')
```

```
(i=1) + (j=1) = 1
(i=1) + (j=2) = 2
(i=1) + (j=3) = 3
===
(i=2) + (j=1) = 2
(i=2) + (j=2) = 4
(i=2) + (j=3) = 6
===
(i=3) + (j=1) = 3
(i=3) + (j=2) = 6
(i=3) + (j=3) = 9
===
```

06. 제어

continue

- 현재 반복에 대해서 루프 내부의 나머지 코드를 건너뛰는데 사용

```
for i in range(10):
    if i == 5:
        continue
    print(i)
```

```
0
1
2
3
4
6
7
8
9
```

06. 제어

break

- 현재 속한 반복문 종료

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

```
0
1
2
3
4
```

06. 제어

while문

- 특정 조건을 만족하는 동안 실행되는 반복문

  while <조건>:

  <들여쓰기> #statement

  <들여쓰기> #statement

```
count = 1

while True:
    print(count)
    count += 1
    if count > 5:
        break
```

```
1
2
3
4
5
```

01 개요
02 자료구조
03 집합형 자료구조
04 문자열
05 연산
06 제어
07 함수
08 패키지와 모듈

07. 함수

함수란?

- 기능 단위로 모듈을 구성
- 코드의 체계화
- 코드의 반복을 피할 수 있고 **코드의 재사용**이 가능

07. 함수

함수 기본구조

```
def 함수명(입력 매개변수):
    <statement>
    <statement>
    <statement>
    return 리턴 값
```

1. 키워드 def는 함수의 시작을 표시
2. 함수 명은 고유하게 식별 가능해야 하고, 식별자 작성 규칙을 따름
3. 함수에 값을 전달하는 입력 매개변수는 선택 사항임
4. 함수 헤더의 끝은 콜론(:)으로 표시
5. 동일한 범위 내 **들여쓰기**가 적용 되어야 함
6. 함수에서 값을 반환하는 return문은 선택 사항

07. 함수

함수의 형태

- return 문은 평가되는 표현식이나, 값을 리턴
- return 문에 표현식이 없거나 return 문 자체가 없으면 함수는 None 객체를 리턴

- 입력 값 (O), return (O)

```
def 함수명(args):
    <statements>
    return <value>
```

- 입력 값 (O), return (X)

```
def 함수명(args):
    <statements>
```

- 입력 값 (X), return (O)

```
def 함수명():
    <statements>
    return <values>
```

- 입력 값 (X), return (X)

```
def 함수명():
    <statements>
```

07. 함수

함수 호출

- **함수 호출 ()**
  - 함수를 정의한 후에는 다른 함수, 프로그램 등에서 호출
  - 함수를 호출하기 위해서는 적절한 인수를 사용하여 호출

```
def sample_function():
    print('함수가 호출 되었습니다!')
```

```
sample_function
```

```
<function __main__.sample_function()>
```

```
sample_function()
```

```
함수가 호출 되었습니다!
```

07. 함수

위치 인수 (positional argument)

- 매개변수 위치에 맞게 값이 전달되는 방식
- 가장 기본적인 매개변수 전달 방식

```
def print_function(a, b, c):
    print(f'a: {a}, b: {b}, c: {c}')
```

```
Print_function(1, 3, 5)
```

```
a: 1, b: 3, c: 5
```

07. 함수

키워드 인수 (keyword argument)

- 함수 호출 시 매개변수의 이름을 명시하여 값을 전달
- 매개변수 이름을 알고 있어야 함
- 순서에 상관 없음

```
def add(a, b, c):
    print(f'a={a}')
    print(f'b={b}')
    print(f'c={c}')
    return a + b + c
```

```
add(1, 4, 4)
```

```
a=1
b=4
C=4
9
```

```
add(1, c=6, b=4)
```

```
a=1
b=4
C=6
11
```

```
add(a=1, c=6, b=5)
```

```
a=1
b=5
c=6
12
```

07. 함수

기본 매개변수 (default parameter)

- 함수 정의시 할당 연산자(=)를 사용하여 매개변수에 기본값을 설정
- 함수 호출시 기본 매개변수의 값을 전달하지 않아도 됨
- 값 전달시 기본값을 덮어 씀
- 함수 정의시 기본값이 없는 매개변수는 기본값 매개변수 뒤에 있을 수 없음

```
def add(a, b=3):
    print(f'a={a}')
    print(f'b={b}')
    return a + b
```

```
add(3)
```

```
a=3
b=3
6
```

```
add(3, 4)
```

```
a=3
b=4
7
```

```
add(5, b=4)
```

```
a=5
b=4
9
```

```
add(a=5, b=4)
```

```
a=5
b=4
9
```

```
add(b=4, a=5)
```

```
a=5
b=4
9
```

```
def add(b=3, a):
    return a + b
```

```
  File "<ipython-input-35-33b02b797714>", line 1
    def add(b=3, a):
            ^
SyntaxError: non-default argument follows default argument
```

07. 함수

가변 매개변수 – tuple

- 매개변수 앞에 *를 붙이면 tuple 형태로 값 전달
- 매개변수의 길이가 가변함 (정해져 있지 않음)

```
def add(*args):
    for arg in args:
        print(arg)
    print('====' * 5)
```

```
add(1)
```

```
1
====================
```

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

```
1
2
3
====================
```

```
add(1, 2, 3, 4, 5, 6, 7)
```

```
1
2
3
4
5
6
7
====================
```

```
def add(a, *args):
    print(f'a={a}')
    for arg in args:
        print(arg)
    print('===='* 5)
```

```
add(1, 2, 3, 4, 5)
```

```
a=1
2
3
4
5
====================
```

```
def add(a, b=2 *args):
    print(f'a={a}')
    print(f'b={b}')
    for arg in args:
        print(arg)
    print('===='*5)
```

```
add(1, 3, 5, 7, 9)
```

```
a=1
b=3
5
7
9
====================
```

07. 함수

가변 매개변수 – dict

- 매개변수 앞에 **를 붙이면 dict 형태로 값 전달
- 매개변수의 길이가 가변함 (정해져 있지 않음)

```
def add(**a):
      for k, v in a.items():
            print(k, v)
      print('====' * 5)
```

```
add(a=1, b=2, c=3)
```

```
a 1
b 2
c 3
====================
```

```
b = {'a':1, 'b':2, 'c':3}
```

```
b
```

```
{'a':1, 'b':2, 'c':3}
```

```
add(**b)
```

```
a 1
b 2
c 3
====================
```

```
def add(a, b=2 *kwargs):
      print(f'a={a}')
      print(f'b={b}')
      print('====' * 5)
      for k, v in kwargs.items():
            print(k, v)
      print('====' * 5)
```

```
add(a=1, b=2, z=3, x=4, y=5)
```

```
a=1
b=2
====================
z 3
x 4
y 5
====================
```

07. 함수

lambda 함수 (익명 함수)

- 이름 없이 정의된 함수
- lambda 키워드를 사용하여 정의
- 여러 개의 인자를 가질 수 있지만 표현식은 단 하나만 가짐
- 람다 함수는 일반적으로 filter(), map()과 같은 내장 함수와 함께 사용

```
a = lambda x: x * 2
```

```
a(4)
```

```
8
```

```
a = lambda x, y: x * y
```

```
a(4, 8)
```

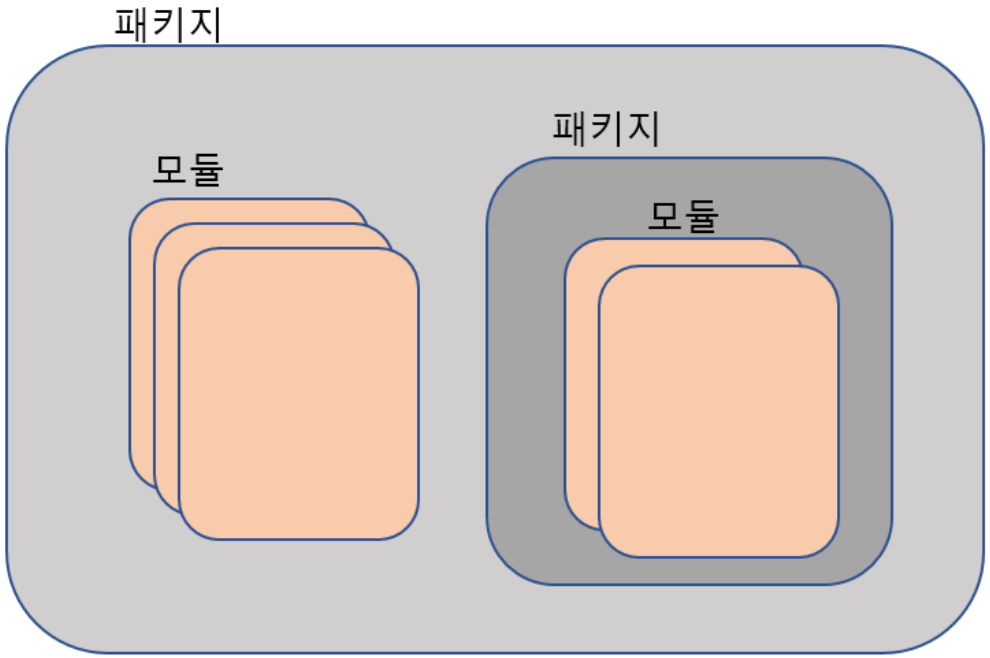
```
32
```

01 개요
02 자료구조
03 집합형 자료구조
04 문자열
05 연산
06 제어
07 함수
08 패키지와 모듈

08. 모듈과 패키지

패키지(package)와 모듈 (module)

- 함수와 모듈 패키지는 아래 그림과 같은 종속관계를 갖는다.
  - 함수가 여러 개 모여 모듈이 되고, 모듈이 여러 개 모여 패키지를 이룬다.
- 패키지는 **폴더**, 모듈은 **파일**의 개념으로 쉽게 생각할 수 있다.




```
└── my_app
    ├── main.py
    ├── package1
    │   ├── module1.py
    │   └── module2.py
    └── package2
        ├── __init__.py
        ├── module3.py
        ├── module4.py
        └── subpackage1
            └── module5.py
```

08. 모듈과 패키지

모듈 (module)

- 재사용 하고자 하는 변수, 함수등을 별도의 파일에 저장하여 다른 Python 파일에서 호출하여 사용
- 파일명이 calculator.py이면 모듈명은 calculator
- 큰 프로그램을 작고 관리하기 쉬운 파일로 나눌 수 있으며, 코드 재사용성이 높아짐
- 자주 사용하는 함수를 정의하고, 이를 다른 파일에서 import해서 사용

```python
# calculator.py

def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    return x / y
```

```python
# main.py
import calculator

result = calculator.add(4, 5)
print('덧셈결과: {}'.format(result))

result = calculator.subtract(4, 5)
print('뺄셈결과: {}'.format(result))

Result = calculator.multiply(4, 5)
print('곱셈결과 : {}'.format(result))

result = calculator.dividd(4, 5)
print('나눗셈결과 : {}'.format(result))
```

└ main.py
└ calculator.py

08. 모듈과 패키지

모듈 사용

- import 모듈명으로 사용할 수 있음
- 함수 단위의 import도 가능

```
import calculator

result = calculator.add(4, 5)
print('덧셈결과: {}'.format(result))
```

```
from calculator import add

result = add(4, 5)
print('덧셈결과: {}'.format(result))
```

08. 모듈과 패키지

alias (별칭)

- 모듈의 이름이 긴 경우, 이를 줄여서 사용함
- 유명한 모듈의 경우 흔히 사용되는 alias가 있고, 이를 암묵적으로 따름

```
import calculator

result = calculator.add(4, 5)
print( ' 덧셈결과: {} ' .format(result))
```

```
from calculator as calc

result = calc.add(4, 5)
print( ' 덧셈결과: {} ' .format(result))
```

```
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sns
```

- numpy: 과학 계산을 위한 패키지
- pandas: 데이터 분석을 할 때 가장 많이 쓰이는 패키지
- matplotlib: 시각화를 위한 패키지
- seaborn: 시각화를 위한 패키지 (matplotlib을 더 쉽게 사용할 수 있도록 도와주는 패키지)

08. 모듈과 패키지

모듈 import 다양한 예제

[참고]
데이터 분석과 관련된 다양한 기능을 제공하는 파이썬 패키지

- import pandas
  - pandas 모듈을 import, pandas로 사용
- import pandas as pd
  - pandas 모듈을 import, pd로 사용
- from pandas import random
  - pandas 모듈 내의 random 모듈만 임포트, random으로 사용(numpy는 사용 불가능)
- from pandas import DataFrame as df
  - pandas 모듈 내의 DataFrame 모듈만 import, df로 사용
- from pandas import *
  - pandas 모듈 내의 모든 것들을 import

교육자

이경록