1.Python Basic

데이터 타입

분류	타입	타입명	리터럴
숫자	정수	int	1, 4, 10
	실수	float	3.156
	복소수	complex	7+7J
	참, 거짓 (True, False)	bool	True
시퀀스	문자열	str	“Hello DongJin”
	리스트	list	[10,20,30,40]
	튜플	tuple	(1,2,3,4)
셋(Set)	셋	Set	{3,5,9,10}
매핑(Mappings)	사전 (딕셔너리)	dict	{‘a’:1, ‘b’:2}

제어문

- if문
- for문
- while문
- break, continue

함수

- 함수 문법
- 매개변수
- 재귀호출
- 람다함수
- 내장/외장함수

내장함수/외장함수

- 내장 함수
  - 내장 함수는 dir(__builtins__)를 통해 조회가 가능
  - dir() : 객체가 가지고 있는 변수나 함수를 보여줌

    >>> dir('hello')
  - enumerate**(**iterable**,** start**=0)** : 순서가 있는 자료형을 입력으로 받아 인덱스를 포함하는 enumerate 객체를 리턴
  - id(object) : 객체를 입력 받아 객체의 고유 주소 값을 리턴
  - len(s) : 입력된 s의 길이를 리턴
  - map(fun, iterable) : 입력 받은 자료형의 각 요소가 함수 fun에 의해 수행된 결과를 묶어서 리턴

    >>> list(map(int, ['1','2','3']))

모듈 사용하기

- 재사용 하고자 하는 변수, 함수, 클래스를 파일에 저장하여 다른 Python 파일에서 호출하여 사용
- 파일명이 myModule.py이면 모듈명은 myModule 임
- 큰 프로그램을 작고 관리하기 쉬운 파일로 나눌 수 있으며, 코드 재 사용성이 높아짐
- 자주 사용하는 함수를 정의하고, 이를 다른 파일에서 import 해서 사용
- 현재 Python 3.x 버전에서는 대략 200개가 넘는 모듈을 지원
  - 문자열(string), 날짜(date), 시간(time), 십진법(decimal), 랜덤(random)
  - 파일(file), os, sqlite3, sys, xml, email, http 등
- 모듈을 사용하는 이유
  - 코드의 재사용으로 인한 유지보수성을 높임
  - 코드를 namespace로 구분하고 관리
  - 복잡하고 어려운 기능을 포함하는 프로그램을 쉽게 구현
  - 모듈 사용

모듈 로딩하기

- 파이썬 모듈 import 방법
  - import 모듈명 [,모듈명...]
  - import os as _os
  - from os import getcwd
  - from os import getcwd as pwd
  - from os import *
- dir() 함수를 이용하여 모듈에 구조(함수, 변수) 확인 가능

```
>>> dir(math)
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh','atan', 'atan2'
  ...
```

모듈 만들기

- 사용자 정의 모듈 생성

```
%%writefile my_calc.py
myname = "홍길동"

def plus(a, b):
    return a+b
def minus(a, b):
    return a-b
```

```
Overwriting my_calc.py
```

- 생성 모듈(my_calc.py)의 사용

```
import my_calc
print(my_calc.plus(3,4))
print(my_calc.minus(3,4))
print(my_calc.myname)
```

```
7
-1
홍길동
```

2.Numpy

Numpy 배열

- Numpy 배열은 동일한 자료형을 가지는 값들이 격자 모양으로 구성됨
  - 각각의 값들은 튜플(양의 정수만) 형태로 색인됨
  - array 함수를 통해 Numpy 배열 생성
  - ndim 함수는 배열의 차원(Rank)를 나타냄
  - shape 함수는 배열의 크기를 튜플 형태로 제공함(행, 열)

- numpy 배열 자료형
  - 다양한 숫자 자료형의 배열 생성 지원 (int32, int64, float32, float64 등)
  - NumPy는 배열이 생성될 때 자료형을 스스로 예측하여 생성함(dtype)

인덱싱과 슬라이싱

```
arr = np.array([[1,2,3,4],
                [5,6,7,8],
                [9,10,11,12]])
arr, arr.shape
```

```
(array([[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]]),
 (3, 4))
```

```
row_r1 = arr[1,:]
row_r1
```

```
array([5, 6, 7, 8])
```

```
row_r2 = arr[1:2, :]
row_r2
```

```
array([[5, 6, 7, 8]])
```

```
col_r1 = a[:, 1]
col_r1
```

```
array([0., 0.])
```

```
col_r2 = a[:, 1:2]
col_r2
```

```
array([[0.],
       [0.]])
```

인덱싱과 슬라이싱(Boolean Indexing)

- Boolean 배열 인덱싱을 통해 배열 속 요소를 선택할 수 있음
- 특정 조건을 만족하는 요소만 선택하고자 할 때 사용

```
arr = np.arange(1,13).reshape(4,3)
arr
```

```
array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
```

```
bool_idx = ( arr >= 5)
bool_idx
```

```
array([[False, False, False],
       [False,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

```
arr[bool_idx]
```

```
array([ 5,  6,  7,  8,  9, 10, 11, 12])
```

```
arr[arr >= 10]
```

```
array([10, 11, 12])
```

유니버셜 함수와 통계함수

- sum함수

```
x = np.array([[1,2],[3,4]])

print(np.sum(x))  # 모든 요소를 합한 값을 연산; 출력 "10"
print(np.sum(x, axis=0))  # 각 열에 대한 합을 연산; 출력 "[4 6]"
print(np.sum(x, axis=1))  # 각 행에 대한 합을 연산; 출력 "[3 7]"
```

```
10
[4 6]
[3 7]
```

- 기본 배열 통계 메서드
  - sum(합), mean(평균), std(표준편차), var(분산), min, max, argmin(최소 원소의 색인값), argmax(최대 원소의 색인값), cumsum(누적합), cumprod(누적곱)
  - 옵션 : axis 값이 0 이면 행단위, 1이면 열 단위로 연산

배열의 연산

- dot함수 : 벡터의 내적, 벡터와 행렬의 곱, 행렬 곱을 위해 '*'대신 'dot'함수사용

```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
```

```
v = np.array([9,10])
w = np.array([11, 12])
```

```
print(x.dot(v))
print(np.dot(x, v))
```

```
[29 67]
[29 67]
```

```
print(x.dot(y))
print(np.dot(x, y))
```

```
[[19 22]
 [43 50]]
[[19 22]
 [43 50]]
```

```
print(v.dot(w))
print(np.dot(v, w))
```

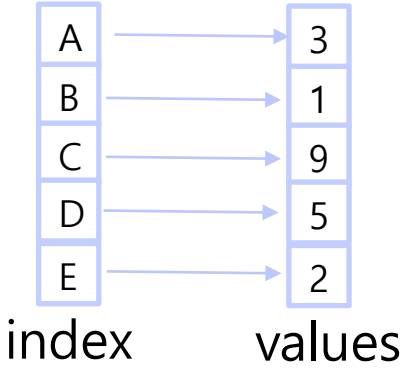
```
219
219
```

3.Pandas

Pandas 자료구조

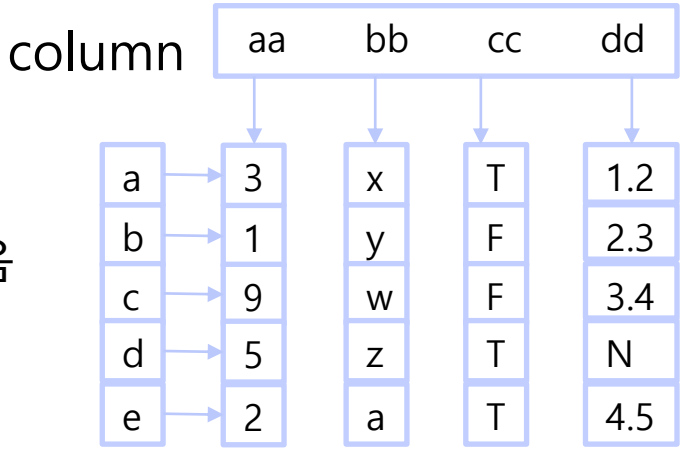
- 데이터구조
  - Series(1D)
  - DataFrame(2D)
  - Panel(3D)

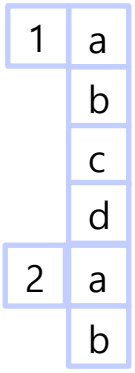
  ✚ Python 기본데이터구조
- index (색인)
  - 모든 데이터의 차원은 각각 인덱스를 가지고 있음
  - join 작업이 가능하고 원하는 데이터 추출가능
- Series
  - numpy.ndarray의 부분집합
  - 일련의 객체를 담을 수 있는 1차원 배열 구조
  - 인덱스 사용가능



Pandas 자료구조

- DataFrame
  - 2차원 데이터 구조
  - 행(row)과 열(column)로 구성됨
  - 각 열은 다른 타입의 데이터를 가질 수 있음
  - 크기변환이 가능(size mutable)
    - 열 추가/삭제 가능
- Hierarchical Indexes
  - 계층 구조를 갖는 인덱스
  - 그룹선택이 용이





데이터 인덱싱, 선택, 필터링

- 데이터선택 방법
  - obj[val] : dataFrame에서 하나의 열 또는 여러 열을 선택

    - boolean 배열 인덱싱, 슬라이싱, boolean dataFrame 사용 가능
  - loc : label 값 기반의 2차원 인덱싱
  - iloc : 정수 기반의 2차원 인덱싱
  - at : label 값 기반의 2차원 인덱싱 (한 개의 값 반환)
  - iat : 정수 기반의 2차원 인덱싱 (한 개의 값 반환)
  - ix : 정수와 label 모두 사용 가능 (deprecated)
- 데이터 삭제
  - drop함수를 이용한 행/열 삭제
  - axis를 기준으로 특정 행 또는 열 삭제 가능

누락데이터 처리

- Pandas는 누락된 데이터를 모두 NaN으로 처리
- Pandas 객체의 모든 기술 통계는 누락된 데이터를 배제하고 처리
- 누락데이터는 Python의 내장 None값 또한 NA 값으로 취급
- NA 처리 관련 함수
  - dropna : 누락된 데이터가 있는 축(행,열)을 제외하며, 어느 정도의 누락 데이터까지 용인할 것인지 지정 가능
  - fillna : 누락된 데이터 값을 대신할 값을 채움
  - isnull : 누락되거나 NA인 값을 알려줌
  - notnull : null이 아닌 값을 알려줌 (NA가 아니면 True)

누락데이터 처리

- 몇 개 이상의 non-na 값이 들어있는 행 또는 열만 살펴보고 싶다면 thresh 옵션값을 설정(주로 시계열 데이터에 사용)

  예) df.dropna(thresh=3) ➔ 3개 이상의 non-na 값이 있는 행만 유지

- 누락된 값 채워 넣기
  - 0으로 채워 넣기 : df1.fillna(0)
  - 평균으로 채워 넣기 : df1.fillna(df.mean())
  - fillna 함수인자

    - value : 비어있는 값 채우기

    - method : 보간 방식 기본적으로 ffill을 사용(forward fill)

    - axis : 값을 채워 넣을 축 (기본값 0)

    - inplace : 호출한 객체에 반영, 기본값을 False

기술통계함수

- 통계적 요약 데이터 보기
  - describe : Series 나 DataFrame의 각 컬럼에 대한 요약통계 표출
    - count, mean, std, min, 25%, 50%, 75%, max
  - var : 표본 분산
  - std : 표준 편차
  - cumsum : 누적 합
  - cumprod : 누적 곱
  - pct_change : 퍼센트 변화율

데이터 변형, 병합 (Concat)

- concat
  - 병합 결과

```
pd.concat([df1, df2, df3])
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

데이터 변형, 병합 (Merge)

- merge 함수는 두 DataFrame의 공통 열(Label) 혹은 인덱스를 기준으로 두 DataFrame을 병합함
- 이 때 기준이 되는 열, 행의 데이터를 키(key)라 부름

	key	lval
0	K1	1
1	K2	2
2	K3	3

	key	rval
0	K0	4
1	K1	5
2	K3	6

```
pd.merge(left, right, on='key')
```

	key	lval	rval
0	K1	1	5
1	K3	3	6

데이터 변형, 병합 (Merge)

- merge 함수는 기본적으로 inner join을 수행하여 교집합 결과를 반환
- how 인자로 left, right, outer을 넘겨 상황에 맞게 병합할 수 있음

  예) pd.merge(df1, df2, how='outer')

```
pd.merge(left, right, on='key', how='right')
```

	key	lval	rval
0	K1	1.0	5
1	K3	3.0	6
2	K0	NaN	4

```
pd.merge(left, right, on='key', how='outer')
```

	key	lval	rval
0	K1	1.0	5.0
1	K2	2.0	NaN
2	K3	3.0	6.0
3	K0	NaN	4.0

```
pd.merge(left, right, on='key', how='inner')
```

	key	lval	rval
0	K1	1	5
1	K3	3	6

데이터 그룹 연산(grouping)

- groupby 함수
  - 특정 컬럼을 index로 지정하여 동일한 index를 가지는 데이터에 대해 여러 집계 함수를 사용 할 수 있음.
  - mean(평균), sum(합), median(중앙값), first(첫번째 값) 등
  - 여러개의 함수를 적용하고 싶은 경우 agg 함수를 사용

	A	B	C	D
0	foo	one	-0.836274	1.512097
1	bar	two	-2.293281	-0.198404
2	foo	one	0.887697	-0.785607
3	bar	three	0.860393	-0.231901

```
df.groupby('A').sum()
```

A	C	D
bar	-1.432888	-0.430304
foo	0.051423	0.726490

```
df.groupby(['A', 'B']).sum()
```

A	B	C	D
bar	three	0.860393	-0.231901
	two	-2.293281	-0.198404
foo	one	0.051423	0.726490

데이터 그룹 연산 (Pivot Tables)

- 피벗 테이블은 데이터에서 하나 이상의 열을 각각 행과 열의 인덱스로 사용하여 데이터를 정렬함

- 피봇 테이블 메서드 옵션
  - values : 집계하려는 열의 Label 혹은 Label의 리스트
  - index : 피벗 테이블 행의 그룹으로 묶을 열의 Label이나 그룹 키
  - columns : 피벗 테이블 열의 그룹으로 묶을 열의 Label 이름이나 그룹 키
  - aggfun : 집계 함수나 함수 리스트, 기본 값은 mean
  - fill_value : 누락된 값을 대체하기 위한 값
  - margins : 부분 합이나 총계를 담기 위한 행/열을 추가할지 여부 (기본 값은 False)

데이터 그룹 연산 (Pivot Tables)

- Pivot Tables 변환을 위한 데이터 불러오기

```
tips.pivot_table(index=['sex', 'day'], columns='smoker',values=['total_bill', 'tip'])
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

		tip		total_bill	
	smoker	Yes	No	Yes	No
sex	day				
Male	Thur	3.058000	2.941500	19.171000	18.486500
	Fri	2.741250	2.500000	20.452500	17.475000
	Sat	2.879259	3.256563	21.837778	19.929063
	Sun	3.521333	3.115349	26.141333	20.403256
Female	Thur	2.990000	2.459600	19.218571	16.014400
	Fri	2.682857	3.125000	12.654286	19.365000
	Sat	2.868667	2.724615	20.266667	19.003846
	Sun	3.500000	3.329286	16.540000	20.824286