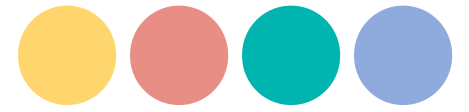


# 넘파이, 판다스 기본개념 익히기



# 파이썬 기본자료형과 컨테이너



## 기본자료형

- 숫자 : 정수형, 실수형
- 불리언(Booleans) : true, false
- 문자열

## 컨테이너

- 리스트
- 딕셔너리
- 집합
- 튜플

# Numpy



출처 : <https://numpy.org/>

- ➡ 고성능 과학(수치)계산을 위한 데이터분석(수치분석) 패키지
  - 벡터 산술연산
  - 다차원 배열 (ndarray)
  - 표준 수학 함수
  - 선형대수, 난수

# Numpy 기술통계(descriptive statistics)



데이터의 개수  
(count)

`len(x)`  
 $x = \{1, 2, 3, 4, 5, \dots, N\} \rightarrow$   
`x = np.array([1, 2, 3, 4, 5, \dots, N])`  
x의 갯수는 numpy에서 `len(x)`

평균  
(average, mean)

`np.mean(x)`

분산  
(variance)

데이터와 샘플 평균간의 거리의 제곱의  
평균이다. 이 값이 작으면 데이터가 모여  
있는 것이다. 반대로 이 값이 크면 흩어져  
있는 것  
`np.var(x)`

# Numpy 기술통계(descriptive statistics)



표준 편차  
(standard deviation)

분산의 제곱근 값이다.  
`np.std(x)`

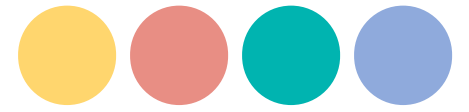
최대값  
(maximum)

데이터 중에서 가장 큰 값  
`np.max(x)`

최소값  
(minimum)

데이터 중에서 가장 작은 값  
`np.min(x)`

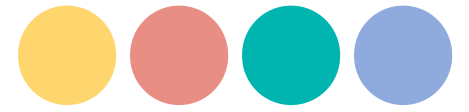
# Numpy 기술통계(descriptive statistics)



중앙값  
(median)

데이터를 크기대로 정렬 하였을 때 가장  
가운데 있는수를 의미한다. 만일  
데이터의 갯수가 짝수인 경우에는 중앙의  
두 수의 평균을 사용  
`np.median(x)`

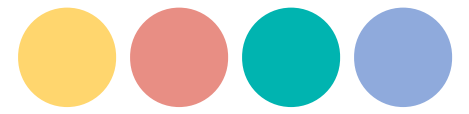
# Numpy 기술통계(descriptive statistics)



## 사분위수 (quartile)

- 데이터를 크기대로 정렬하였을 때 1/4, 2/4, 3/4 위치에 있는 수를 말한다.
- 1사분위수, 2사분위수, 3사분위수라 한다.
- 데이터가 100개가 있다면 25번째 순서가 1사분위수 이다.
- 2사분위수는 중앙값과 같다.
- 위치를 1/100단위로 나눈 분위수(percentile)를 사용하기도 함.
- 1사분위수는 25% 백분위수와 같다.

# Numpy 기술통계(descriptive statistics)



## 사분위수 (quartile)

<code>np.percentile(x, 0)</code>	#최솟값
<code>np.percentile(x, 25)</code>	#1사분위수
<code>np.percentile(x, 50)</code>	#2사분위수
<code>np.percentile(x, 75)</code>	#3사분위수
<code>np.percentile(x, 100)</code>	#최댓값

## sum

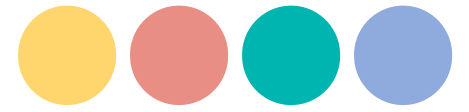
여러개의 수를 연속하여 더하는 연산  
`np.sum(x)`

## product

여러개의 수를 연속하여 곱하는 연산  
`np.product(x)`



# Numpy 배열



 같은 종류의 데이터를 담는 자료형 (동일한 타입의 값)

**rank**

배열 차원 (ndim)

**shape**

각 차원 크기를  
튜플로 리턴(행, 열)

**dtype**

배열에 저장된  
자료를 알려주는  
객체

# Numpy 다차원배열

 ndarray

다차원 배열 객체

**array()**

배열 생성

**reshape()**

배열 모양 변경

**ones()**

1이 들어있는  
배열 생성

**eye()**

단위 행렬 생성

**arange()**

배열 생성 시  
지정 값까지  
자동 생성

**zeros()**

0이 들어있는  
배열 생성

**full()**

모든 값이 특정  
상수인 배열을  
생성

# Numpy Cheat Sheet



Course Outline

EXERCISE

### Encoding time by color

The screen only has two dimensions, but we can encode another dimension in the scatter plot using color. Here, we will visualize the `climate_change` dataset, plotting a scatter plot of the `"co2"` column, on the x-axis, against the `"relative_temp"` column, on the y-axis. We will encode time using the color dimension, with earlier times appearing as darker shades of blue and later times appearing as brighter shades of yellow.

INSTRUCTIONS

100 XP

- Using the `ax.scatter` method add a scatter plot of the `"co2"` column (x-axis) against the `"relative_temp"` column.
- Use the `c` key-word argument to pass in the index of the DataFrame as input to color each point according to its date.
- Set the x-axis label to `"CO2 (ppm)"` and the y-axis label to `"Relative temperature (C)"`.

Take Hint (-30 XP)

query.sql

```
1 fig, ax = plt.subplots()
2
3 # Add data: "co2", "relative_temp" as
4 x-y, index as color
5 ax.scatter(climate_change["co2"],
6           climate_change["relative_temp"],
7           c=climate_change.index)
8
9 # Set the x-axis label to "CO2 (ppm)"
10 ax.set_xlabel("CO2 (ppm)")
11
12 # Set the y-axis label to "Relative
13 temperature (C)"
14 ax.set_ylabel("Relative temperature
15 (C)")
16
```

Run Code

Submit Answer

Plots

IPython Shell

In [1]:

출처 : <https://www.datacamp.com/community/data-science-cheatsheets>

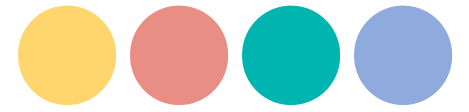
# Pandas



출처 : <https://pandas.pydata.org/>

- ➡ pandas는 데이터 조작 및 분석을 위해 Python 프로그래밍 언어로 작성된 **소프트웨어 라이브러리**
- ➡ CSV, Excel, JSON 등의 데이터를 읽고 **원하는 데이터 형식으로 변환**
- ➡ Series, Dataframe

# Pandas 데이터 입출력 도구



File Format	Reader	Writer
CSV	read_csv	to_csv
JSON	read_json	to_json
HTML	read_html	to_html
local clipboard	read_clipboard	to_clipboard
MS Excel	read_excel	to_excel
HDF5 Format	read_hdf	to_hdf
SQL	read_sql	to_sql

출처 : <http://pandas.pydata.org>

# Pandas Cheat Sheet

## Data Wrangling

with pandas  
Cheat Sheet

<http://pandas.pydata.org>

### Syntax - Creating DataFrames

```
df = pd.DataFrame({
    "a": [4, 5, 6],
    "b": [7, 8, 9],
    "c": [10, 11, 12]},
    index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame([
    [4, 7, 10],
    [5, 8, 11],
    [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

```
df = pd.DataFrame({
    "a": [4, 5, 6],
    "b": [7, 8, 9],
    "c": [10, 11, 12]},
    index = pd.MultiIndex.from_tuples(
        [('d',1), ('d',2), ('e',2)],
        names=['n', 'v']))
```

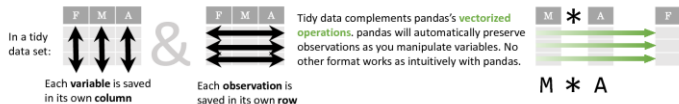
Create DataFrame with a MultiIndex

### Method Chaining

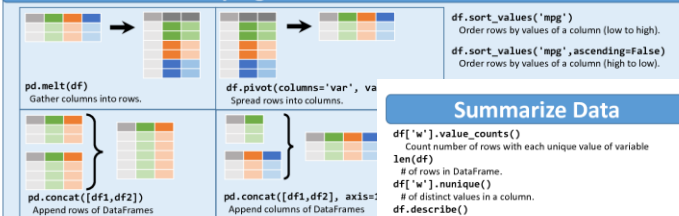
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
      .rename(columns={
          'variable': 'var',
          'value': 'val'})
      .query('val > 200'))
```

## Tidy Data - A foundation for wrangling in pandas



### Reshaping Data - Change the layout of a data set



### Summarize Data

```
df['w'].value_counts()
Count number of rows with each unique value of variable

len(df)
# of rows in DataFrame.

df['w'].nunique()
# of distinct values in a column.

df.describe()
Basic descriptive statistics for each column (or GroupBy)
```

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

```
sum()
Sum values of each object.

count()
Count non-NA/null values of each object.

median()
Median value of each object.

quantile([0.25, 0.75])
Quantiles of each object.

std()
Standard deviation of each object.

apply(function)
Apply function to each object.
```

### Group Data

```
df.groupby(by="col")
Return a GroupBy object, grouped by values in column named "col".

df.groupby(level="ind")
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group.
Additional GroupBy functions:

size()
Size of each group.

agg(function)
Aggregate group using function.
```

### Windows

```
df.expanding()
Return an Expanding object allowing summary functions to be applied cumulatively.

df.rolling(n)
Return a Rolling object allowing summary functions to be applied to windows of length n.
```

### Handling Missing Data

```
df.dropna()
Drop rows with any column having NA/null data.

df.fillna(value)
Replace all NA/null data with value.
```

### Make New Columns

```
df.assign(Area=lambda df: df.Length*df.Height)
Compute and append one or more new columns.

df['Volume'] = df.Length*df.Height*df.Depth
Add single column.

pd.qcut(df.col, n, labels=False)
Bin column into n buckets.
```

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

```
max(axis=1)
Element-wise max.

clip(lower=-10, upper=10)
Trim values at input thresholds.

min(axis=1)
Element-wise min.

abs()
Absolute value.
```

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

```
shift(1)
Copy with values shifted by 1.

rank(method='dense')
Ranks with no gaps.

rank(method='min')
Ranks. Ties get min rank.

rank(pct=True)
Ranks rescaled to interval [0, 1].

rank(method='first')
Ranks. Ties go to first value.
```

### Plotting

```
df.plot.hist()
Histogram for each column.

df.plot.scatter(x='w', y='h')
Scatter chart using pairs of points.
```

### Combine Data Sets

```
adf + bdf
Join matching rows from adf to bdf.
```

```
pd.merge(adf, bdf,
         how='left', on='x1')
Join matching rows from adf to bdf.
```

```
pd.merge(adf, bdf,
         how='right', on='x1')
Join matching rows from bdf to adf.
```

```
pd.merge(adf, bdf,
         how='inner', on='x1')
Join data. Retain only rows in both sets.
```

```
pd.merge(adf, bdf,
         how='outer', on='x1')
Join data. Retain all values, all rows.
```

Filtering Joins

```
adf[adf.x1.isin(bdf.x1)]
All rows in adf that have a match in bdf.
```

```
adf[~adf.x1.isin(bdf.x1)]
All rows in adf that do not have a match in bdf.
```

```
ydf + zdf
Rows that appear in both ydf and zdf (Intersection).
```

Set-like Operations

```
pd.merge(ydf, zdf,
         indicator=True)
Rows that appear in either or both ydf and zdf (Union).
```

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('merge == "left_only"')
.drop(columns=['_merge'])
Rows that appear in ydf but not zdf (Setdiff).
```