# 기본 자료 처리


- **<차근차근 파이썬 코딩실습 - 기본편> 36 쪽**부터,
- 30 분: 기본 자료 처리에 대해 소개
- 60 분: 질의응답

# 강사 소개

**Kwan-Gu, Baek**
Kwan-Gu

**@python @Sql** @SpatialInformation
@AtmosphericScience **@datascience**
**@MachineLearning**
**@SeoulNationalUniversity**

- 이름(메일): **백관구(bouguereau1130@gmail.com)**
- 소속: 서울대 지구환경과학부 대기과학전공 석박통합과정 4 년
- 분야: 머신러닝(딥러닝) | 공간정보학(고해상도 자료 생산 알고리즘)
- 언어: **파이썬** (5 년차)

# 목차



# 1-A. 파이썬 기본 함수로 ASCII 파일 읽기

| | |
|---|---|
| **open(파일 경로)** | • 파일 경로를 불러와 열기<br>• Default: read "r" 모드 |
| **readlines()**<br>**read().splitlines()** | • 한 줄 한 줄씩 모든 줄을 list에 저장 |
| **strip()** | • 문자열 양쪽 끝에 있는 공백, ₩n(줄바꿈 기호) 제거 |
| **split(구분자)** | • 구분자(예: ","  또는 " " 등)를 기준으로 문자열을 분리 |

## (예제)

In [1]:

```
# stash.csv 파일 다운로드: STASH 코드(기후모델 내 변수를 정의하는 이름)
!curl --location http://reference.metoffice.gov.uk/um/stash?_format=csv > stash.csv
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0
100 44884    0 44884    0     0  18083      0 --:--:--  0:00:02 --:--:-- 18076
100  340k    0  340k    0     0   103k      0 --:--:--  0:00:03 --:--:--  103k
100 2253k    0 2253k    0     0   512k      0 --:--:--  0:00:04 --:--:--  512k
100 7404k    0 7404k    0     0  1408k      0 --:--:--  0:00:05 --:--:-- 1496k
```

In [2]:

```
with open("stash.csv") as f:
    lines = f.readlines()

len(lines)
```

Out[2]:

5463

In [3]:

```
print(lines[0])
```

```
<http://reference.metoffice.gov.uk/um/c4/stash/blev>,<http://reference.metoffice.g
ov.uk/um/c4/stash/cfff>,<http://reference.metoffice.gov.uk/um/c4/stash/cfll>,<htt
p://reference.metoffice.gov.uk/um/c4/stash/dataT>,<http://reference.metoffice.gov.
uk/um/c4/stash/dumpP>,<http://reference.metoffice.gov.uk/um/c4/stash/grid>,<htt
p://reference.metoffice.gov.uk/um/c4/stash/halo>,<http://reference.metoffice.gov.u
k/um/c4/stash/item>,<http://reference.metoffice.gov.uk/um/c4/stash/lbvc>,<http://r
eference.metoffice.gov.uk/um/c4/stash/levCom>,<http://reference.metoffice.gov.uk/u
m/c4/stash/levelF>,<http://reference.metoffice.gov.uk/um/c4/stash/levelL>,<http://
reference.metoffice.gov.uk/um/c4/stash/levelT>,<http://reference.metoffice.gov.uk/
um/c4/stash/model>,<http://reference.metoffice.gov.uk/um/c4/stash/name>,<http://re
ference.metoffice.gov.uk/um/c4/stash/option_code>,<http://reference.metoffice.gov.
uk/um/c4/stash/pc1>,<http://reference.metoffice.gov.uk/um/c4/stash/pc2>,<http://re
ference.metoffice.gov.uk/um/c4/stash/pc3>,<http://reference.metoffice.gov.uk/um/c
4/stash/pc4>,<http://reference.metoffice.gov.uk/um/c4/stash/pc5>,<http://referenc
e.metoffice.gov.uk/um/c4/stash/pc6>,<http://reference.metoffice.gov.uk/um/c4/stas
h/pc7>,<http://reference.metoffice.gov.uk/um/c4/stash/pc8>,<http://reference.metof
fice.gov.uk/um/c4/stash/pc9>,<http://reference.metoffice.gov.uk/um/c4/stash/pcA>,<
http://reference.metoffice.gov.uk/um/c4/stash/point>,<http://reference.metoffice.g
ov.uk/um/c4/stash/ppfc>,<http://reference.metoffice.gov.uk/um/c4/stash/pseudF>,<ht
tp://reference.metoffice.gov.uk/um/c4/stash/pseudL>,<http://reference.metoffice.go
v.uk/um/c4/stash/pseudT>,<http://reference.metoffice.gov.uk/um/c4/stash/rblevv>,<h
ttp://reference.metoffice.gov.uk/um/c4/stash/rotate>,<http://reference.metoffice.g
ov.uk/um/c4/stash/section>,<http://reference.metoffice.gov.uk/um/c4/stash/space>,<
http://reference.metoffice.gov.uk/um/c4/stash/time>,<http://reference.metoffice.go
v.uk/um/c4/stash/tlev>,<http://reference.metoffice.gov.uk/um/c4/stash/user>,<htt
p://reference.metoffice.gov.uk/um/c4/stash/version_mask>,@id,rdf:type,rdfs:label,s
kos:notation
```

In [4]:

```
print(lines[1])
```

```
0,<http://reference.metoffice.gov.uk/um/met08code/12>,<http://reference.metoffice.
gov.uk/um/met08code/9999>,<http://reference.metoffice.gov.uk/um/c4/data_type_code/
1>,<http://reference.metoffice.gov.uk/um/f3/lbpack_code/2>,<http://reference.metof
fice.gov.uk/um/c4/grid_code/1>,"",1,<http://reference.metoffice.gov.uk/um/fieldcod
e/129>,<http://reference.metoffice.gov.uk/um/c4/levcom_code/0>,<http://reference.m
etoffice.gov.uk/um/c4/end_level_code/-1>,<http://reference.metoffice.gov.uk/um/c4/
end_level_code/-1>,<http://reference.metoffice.gov.uk/um/c4/level_type_code/5>,<ht
tp://reference.metoffice.gov.uk/um/c4/model_code/1>,PSTAR AFTER TIMESTEP,000000000
00000000000,0,0,0,0,0,16,-99,-99,-99,-99,0,<http://reference.metoffice.gov.uk/um/f
ieldcode/8>,<http://reference.metoffice.gov.uk/um/c4/first_pseudolevel_code/0>,<ht
tp://reference.metoffice.gov.uk/um/c4/last_pseudolevel_code/0>,<http://reference.m
etoffice.gov.uk/um/c4/pseudolevel_type_code/0>,0,<http://reference.metoffice.gov.u
k/um/c4/rotate_code/0>,0,<http://reference.metoffice.gov.uk/um/c4/space_code/2>,<h
ttp://reference.metoffice.gov.uk/um/c4/time_code/1>,0,0,0000000000000000001,<htt
p://reference.metoffice.gov.uk/um/stash/m01s00i001>,<http://reference.metoffice.go
v.uk/um/c4/stash/Stash>|skos:Concept,PSTAR AFTER TIMESTEP,m01s00i001
```

In [5]:

```
type(lines[10])
```

Out[5]:

```
str
```

In [6]:

```
stash_dict = {} # 빈 딕셔너리 선언

for line in lines:
    words = line.strip().split(",")
    stash_dict[words[-1]] = words[-2]

print(len(stash_dict))
```

```
5463
```

In [7]:

```
for i in range(5):
    print(list(stash_dict.items())[i])
```

```
('skos:notation', 'rdfs:label')
('m01s00i001', 'PSTAR AFTER TIMESTEP')
('m01s00i002', 'U COMPNT OF WIND AFTER TIMESTEP')
('m01s00i003', 'V COMPNT OF WIND AFTER TIMESTEP')
('m01s00i004', 'THETA AFTER TIMESTEP')
```

In [8]:

```
while True:
    target_val = input(">> Input: ")
    if not target_val: break

    for key in stash_dict.keys():
        name = stash_dict[key]
        if (name.find(target_val) != -1) | (key.find(target_val) != -1):
            print(f"Found {target_val} at key {key} [{stash_dict[key]}]")
```

```
>> Input: m01s00i001
Found m01s00i001 at key m01s00i001 [PSTAR AFTER TIMESTEP]
>> Input: V COMPNT OF WIND AFTER TIMESTEP
Found V COMPNT OF WIND AFTER TIMESTEP at key m01s00i003 [V COMPNT OF WIND AFTER TI
MESTEP]
Found V COMPNT OF WIND AFTER TIMESTEP at key m01s21i003 [CM1: V COMPNT OF WIND AFT
ER TIMESTEP]
Found V COMPNT OF WIND AFTER TIMESTEP at key m01s22i003 [CM2: V COMPNT OF WIND AFT
ER TIMESTEP]
Found V COMPNT OF WIND AFTER TIMESTEP at key m01s23i003 [CM3: V COMPNT OF WIND AFT
ER TIMESTEP]
Found V COMPNT OF WIND AFTER TIMESTEP at key m01s24i003 [CM4: V COMPNT OF WIND AFT
ER TIMESTEP]
>> Input: V COMPNT
Found V COMPNT at key m01s00i003 [V COMPNT OF WIND AFTER TIMESTEP]
Found V COMPNT at key m01s00i203 [V COMPNT PERTURBATION- DUMMY  (ret)]
Found V COMPNT at key m01s03i003 [V COMPNT OF WIND AFTER B.LYR (retd)]
Found V COMPNT at key m01s06i003 [V COMPNT OF WIND AFTER G.WAVE DRAG]
Found V COMPNT at key m01s07i003 [V COMPNT OF WIND AFTER VERT DIFSION]
Found V COMPNT at key m01s10i003 [V COMPNT OF WIND AFTER ADJUSTMENT]
Found V COMPNT at key m01s21i003 [CM1: V COMPNT OF WIND AFTER TIMESTEP]
Found V COMPNT at key m01s22i003 [CM2: V COMPNT OF WIND AFTER TIMESTEP]
Found V COMPNT at key m01s23i003 [CM3: V COMPNT OF WIND AFTER TIMESTEP]
Found V COMPNT at key m01s24i003 [CM4: V COMPNT OF WIND AFTER TIMESTEP]
Found V COMPNT at key m01s30i002 [V COMPNT OF WIND RHO GRID]
Found V COMPNT at key m01s30i202 [V COMPNT OF WIND ON P LEV/UV GRID]
Found V COMPNT at key m01s35i002 [V COMPNT OF WIND AFTER SKEB2]
Found V COMPNT at key m01s35i004 [V COMPNT OF WIND INCR SKEB2]
Found V COMPNT at key m01s35i006 [ROT V COMPNT OF WIND INCR SKEB2]
Found V COMPNT at key m01s35i008 [DIV V COMPNT OF WIND INCR SKEB2]
Found V COMPNT at key m01s39i012 [V COMPNT OF WIND AFTER NUDGING]
>> Input:
```

---

# 1-B. NumPy 패키지로 ASCII 파일 읽기

| | |
|---|---|
| **import numpy as np** | • NumPy 패키지(1차원 또는 다차원 배열 array 연산에 주로 사용) |
| **np.genfromtxt(**파일 경로,<br>encoding=인코딩 타입,<br>dtype=데이터 타입,<br>delimiter=구분자,<br>names=("변수명1", "변수명2", ...),<br>skip_header=건너뛸 줄의 개수**)** | • 주어진 파일 경로에서 배열을 불러옴(numpy.ndarray 형태로 생성) |

## (1) 슬라이싱(Slicing): 전체 배열에서 사용하고자 하는 부분만 추출

| | |
|---|---|
| **np.where**(조건) | • 주어진 조건을 만족하는 인덱스를 반환 |

## (2) 통계기법

| | | |
|---|---|---|
| **np.min**(ndarray) | **ndarray.min()** | • 최소값<br>• 배열에 결측 nan이 있으면 nan을 출력 |
| **np.nanmin**(ndarray) | **ndarray.nanmin()** | • 결측을 제외한 최소값<br>• 결측 nan을 자동으로 제외하고 최소값 출력 |
| **np.max**(ndarray)<br>**np.nanmax**(ndarray) | **ndarray.max()**<br>**ndarray.nanmax()** | • 최대값 |
| **np.mean**(ndarray)<br>**np.nanmean**(ndarray) | **ndarray.mean()**<br>**ndarray.nanmean()** | • 평균값 |
| **np.std**(ndarray)<br>**np.nanstd**(ndarray) | **ndarray.std()**<br>**ndarray.nanstd()** | • 표준편차 |
| **np.sum**(ndarray)<br>**np.nansum**(ndarray) | **ndarray.sum()**<br>**ndarray.nansum()** | • 합계 |

## (3) 자료 저장(ASCII)

| | |
|---|---|
| **np.savetxt**(파일 경로, 저장할 배열, fmt = 저장 포맷, delimiter = 구분자) | • ASCII 파일로 저장 |

## (4) 자료 저장(Binary)

| | |
|---|---|
| **f = open(파일 경로, "wb")** | • Binary 파일을 생성하고 열기<br>• "w": 쓰기 모드, "b": Binary |
| **ndarray.tofile**(f) | • 생성한 파일에 배열 입력 |
| **f.close()** | • 생성한 파일을 닫고 저장 |

## (예제)

- "KMA_seoul_raw_data.csv" 파일

```
1
2 미세먼지농도
3 [검색조건]
4 지점명 : 서울, 자료구분:월자료 , 기간 : 2008년01월 ~ 2020년 월
5
6 지점번호,지점명,일자,미세먼지농도(㎍/㎥)
7 108,서울,2008-04,
8 108,서울,2008-05,63
9 108,서울,2008-06,42
10 108,서울,2008-07,37
```

In [9]:

```
import numpy as np
print(np.__version__)
```

1.19.1

In [10]:

```
fname = "KMA_seoul_raw_data.csv"
seoul = np.genfromtxt(fname, encoding = "euc-kr", dtype = None, delimiter = ",", names = ("loc_num", "loc_name", "date", "conc"),W
                      skip_header = 6)

print(seoul.shape)
```

```
(142,)
```

In [11]:

```
print(seoul[: 5])
```

```
[(108, '서울', '2008-04', -1) (108, '서울', '2008-05', 63)
 (108, '서울', '2008-06', 42) (108, '서울', '2008-07', 37)
 (108, '서울', '2008-08', 27)]
```

In [12]:

```
seoul_range = seoul[np.where(seoul["date"] == "2009-01")[0][0] : np.where(seoul["date"] == "2010-12")[0][0] + 1]

print(seoul_range)
```

```
[(108, '서울', '2009-01', 57) (108, '서울', '2009-02', 81)
 (108, '서울', '2009-03', 59) (108, '서울', '2009-04', 63)
 (108, '서울', '2009-05', 56) (108, '서울', '2009-06', 47)
 (108, '서울', '2009-07', 39) (108, '서울', '2009-08', 31)
 (108, '서울', '2009-09', 39) (108, '서울', '2009-10', 49)
 (108, '서울', '2009-11', 43) (108, '서울', '2009-12', 68)
 (108, '서울', '2010-01', 56) (108, '서울', '2010-02', 47)
 (108, '서울', '2010-03', 60) (108, '서울', '2010-04', 46)
 (108, '서울', '2010-05', 55) (108, '서울', '2010-06', 48)
 (108, '서울', '2010-07', 30) (108, '서울', '2010-08', 33)
 (108, '서울', '2010-09', 27) (108, '서울', '2010-10', 38)
 (108, '서울', '2010-11', 69) (108, '서울', '2010-12', 64)]
```

In [13]:

```
print(seoul_range["conc"])
```

```
[57 81 59 63 56 47 39 31 39 49 43 68 56 47 60 46 55 48 30 33 27 38 69 64]
```

In [14]:

```
print(type(seoul_range))
```

```
<class 'numpy.ndarray'>
```

In [15]:

```
print(seoul_range["conc"].min())
```

```
27
```

In [16]:

```
print(seoul_range["conc"].max())
```

81

In [17]:

```
print(seoul_range["conc"].mean())
```

50.208333333333336

In [18]:

```
print(seoul_range["conc"].std())
```

13.524604635831524

In [19]:

```
print(seoul_range["conc"].sum())
```

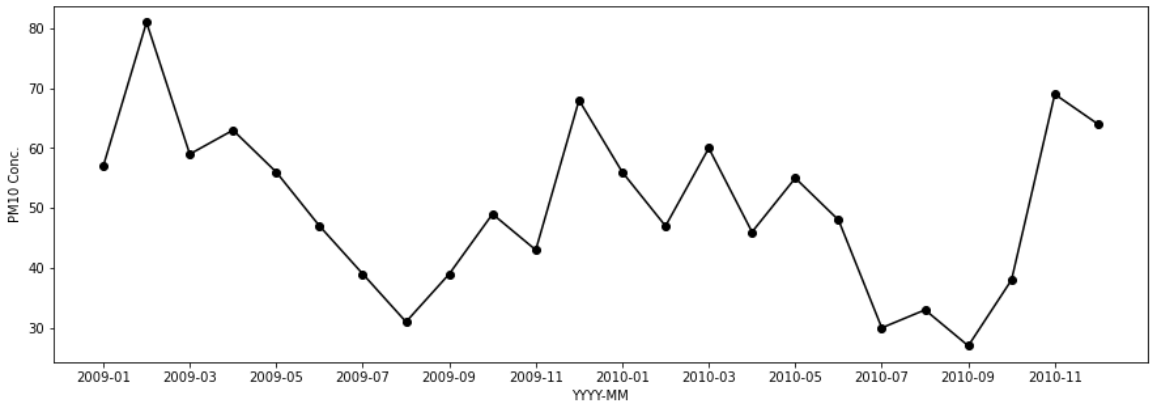
1205

In [20]:

```
import matplotlib.pyplot as plt

plt.figure(figsize = (15, 5))
plt.plot(seoul_range["date"], seoul_range["conc"], c = "k", marker = "o")
plt.xticks(seoul_range["date"][:: 2])
plt.xlabel("YYYY-MM")
plt.ylabel("PM10 Conc.")
```

Out[20]:

Text(0, 0.5, 'PM10 Conc.')




In [21]:

```
np.savetxt("seoul_pm_values.csv", [seoul_range["conc"]], fmt = "%.1f", delimiter = ",")
```

In [22]:

```
f = open("seoul_pm_values.bin", "wb")
seoul_range["conc"].tofile(f)
f.close()
```

# 2-A. Binary (.bin)

## (1) Binary 파일 읽기

| | |
|---|---|
| **np.fromfile(**파일 경로, dtype = 자료형**)** | • Binary 파일 읽기<br>• NumPy 자료형 참고 |

## (2) NumPy 자료형

| Numpy type | C type | Description |
|---|---|---|
| *np.int8* | `int8_t` | Byte (-128 to 127) |
| *np.int16* | `int16_t` | Integer (-32768 to 32767) |
| *np.int32* | `int32_t` | Integer (-2147483648 to 2147483647) |
| *np.int64* | `int64_t` | Integer (-9223372036854775808 to 9223372036854775807) |
| *np.uint8* | `uint8_t` | Unsigned integer (0 to 255) |
| *np.uint16* | `uint16_t` | Unsigned integer (0 to 65535) |
| *np.uint32* | `uint32_t` | Unsigned integer (0 to 4294967295) |
| *np.uint64* | `uint64_t` | Unsigned integer (0 to 18446744073709551615) |
| *np.intp* | `intptr_t` | Integer used for indexing, typically the same as `ssize_t` |
| *np.uintp* | `uintptr_t` | Integer large enough to hold a pointer |
| *np.float32* | `float` | |
| *np.float64* / *np.float_* | `double` | Note that this matches the precision of the builtin python *float*. |
| *np.complex64* | `float complex` | Complex number, represented by two 32-bit floats (real and imaginary components) |
| *np.complex128* / *np.complex_* | `double complex` | Note that this matches the precision of the builtin python *complex*. |

## (예제)

- NSIDC **Sea Ice Concentrations** from Nimbus-7 SMMR and DMSP SSM/I-SSMIS Passive Microwave Data, Version 1 (https://nsidc.org/data/nsidc-0051 (https://nsidc.org/data/nsidc-0051))

Data files may contain integers from 0 to 255, as described in Table 1.

**Table 1.** Description of Data Values

| Data Value | Description |
|---|---|
| 0 - 250 | Sea ice concentration (fractional coverage scaled by 250) |
| 251 | Circular mask used in the Arctic to cover the irregularly-shaped data gap around the pole (caused by the orbit inclination and instrument swath) |
| 252 | Unused |
| 253 | Coastlines |
| 254 | Superimposed land mask |
| 255 | Missing data |

In [23]:

```
ice = np.fromfile("nt_20180731_f17_v1.1_n.bin", dtype = "uint8") # unsigned integer 8-bit

print(ice.shape)
```

(136492,)

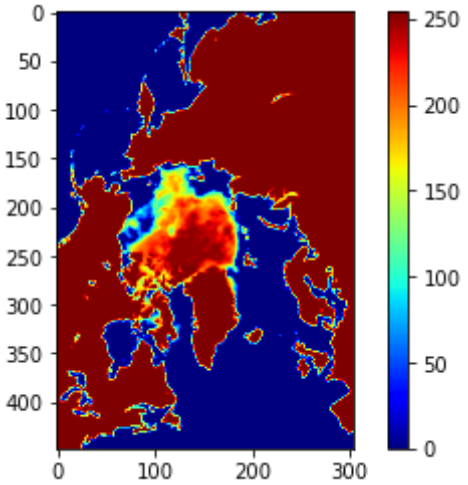
In [24]:

```
ice = ice[300 :] # 300-byte descriptive header
ice = ice.reshape((448, 304)) # np.reshape() 또는 ndarray.reshape()

print(ice.shape)
```

(448, 304)

In [25]:

```
plt.imshow(ice, cmap = "jet") # colormap: https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html
plt.colorbar()

plt.show()
```



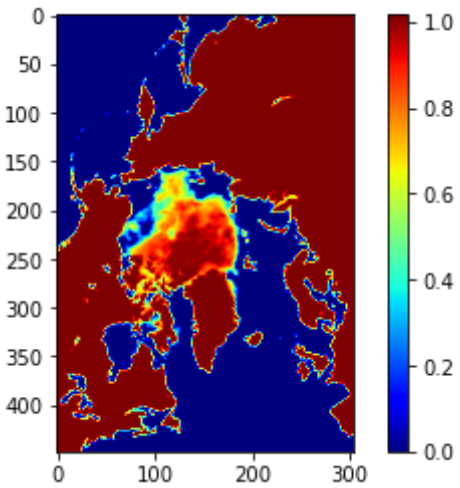

In [26]:

```
ice_frac = ice / 250. # Fractional parameter 로 변환

print(ice_frac)
```

```
[[0.    0.    0.    ... 1.016 1.016 1.016]
 [0.    0.    0.    ... 1.016 1.016 1.016]
 [0.    0.    0.    ... 1.016 1.016 1.016]
 ...
 [1.016 1.016 1.016 ... 0.    0.    0.   ]
 [1.016 1.016 1.016 ... 0.    0.    0.   ]
 [1.016 1.016 1.016 ... 0.    0.    0.   ]]
```

In [27]:

```
plt.imshow(ice_frac, cmap = "jet")
plt.colorbar()

plt.show()
```



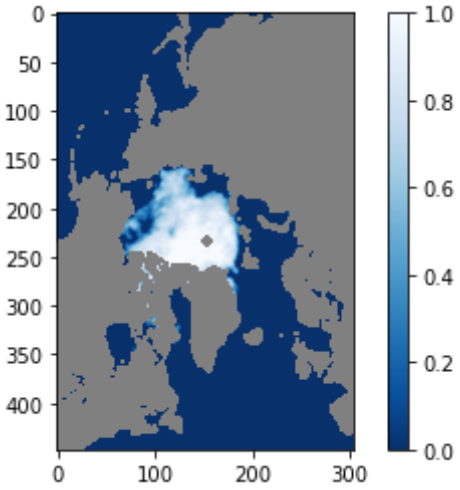

In [28]:

```
ice_masked = np.ma.masked_greater(ice_frac, 1.) # 1보다 큰 값에 mask
"""
np.ma.masked_greater_equal()
np.ma.masked_equal()
np.ma.masked_less_equal()
np.ma.masked_less()
"""
print(ice_masked)
```

```
[[0.0 0.0 0.0 ... -- -- --]
 [0.0 0.0 0.0 ... -- -- --]
 [0.0 0.0 0.0 ... -- -- --]
 ...
 [-- -- -- ... 0.0 0.0 0.0]
 [-- -- -- ... 0.0 0.0 0.0]
 [-- -- -- ... 0.0 0.0 0.0]]
```

In [29]:

```
cm = plt.cm.get_cmap("Blues_r")
cm.set_bad(color = "grey") # mask 부분을 회색으로 표시

plt.imshow(ice_masked, cmap = cm)
plt.colorbar()

plt.show()
```




# 2-B. MATLAB (.mat)

## (1) MATLAB 파일 읽기 & 저장

| | |
|---|---|
| **from scipy.io import loadmat, savemat** | • SciPy 패키지의 .mat 파일 읽기 & 저장 |
| **savemat**(파일 경로, {키: 데이터}) | • 파일을 만들어 키(key)와 해당하는 데이터를 저장 |
| **mat = loadmat**(파일 경로) | • 파일 경로의 .mat 파일을 읽기 |
| **mat.keys()** | • 파일 안에 있는 키(변수명) 확인 |
| **mat[키]** | • 파일 안에 있는 데이터 불러오기 |

## (예제)

In [30]:

```
from scipy.io import loadmat, savemat

savemat("My_ice_fraction_20180731_448x304.mat", {"ice_frac": ice_frac})
```

In [31]:

```
all_variables = loadmat("My_ice_fraction_20180731_448x304.mat")

print(all_variables.keys())
```

```
dict_keys(['__header__', '__version__', '__globals__', 'ice_frac'])
```

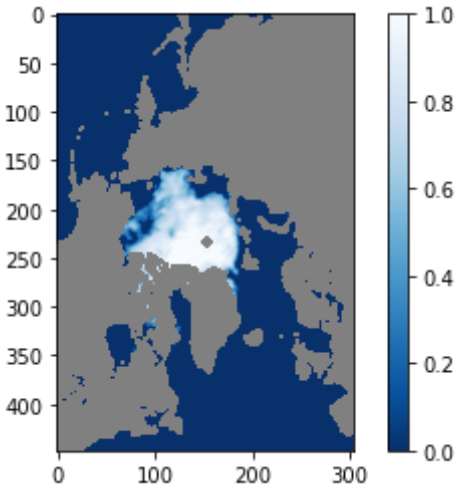
In [32]:

```
ice_frac = all_variables["ice_frac"]

print(ice_frac.shape)
```

```
(448, 304)
```

In [33]:

```
ice_masked = np.ma.masked_greater(ice_frac, 1.)

plt.imshow(ice_masked, cmap = cm)
plt.colorbar()

plt.show()
```




---

# 2-C. NetCDF4 (.nc)

## (1) NetCDF4 파일 읽기

| | |
|---|---|
| **from netCDF4 import Dataset** | • netCDF4 패키지 |
| **nc = Dataset(파일 경로)** | • NetCDF4 파일 읽기<br>• Default: "r" read 모드 |
| **nc.variables** | • NetCDF4 파일의 변수 정보 확인 |
| **nc.variables[변수명]** | • 변수를 netCDF4 클래스 변수로 저장 |
| **nc.variables[변수명][:]** | • 변수를 Numpy ndarray 배열로 저장 |
| **nc.close()** | • 파일 닫기 |

## (2) NetCDF4 파일 저장

| | |
|---|---|
| **nc = Dataset(파일 경로, "w")** | • .nc 파일 생성 |
| **nc.createDimension(차원 이름, 크기)** | • 차원 생성 |
| **var = nc.createVariable(변수명, 자료형, 차원)** | • 변수 생성 |
| **var[:] = array** | • 생성한 변수에 값 입력 |
| **var.속성 이름 = 속성 내용** | • 생성한 변수의 속성을 저장 |
| **nc.close()** | • 생성한 파일을 닫고 저장 |

## (예제) NetCDF4 파일 읽기

In [34]:

```
from netCDF4 import Dataset

nc_file = Dataset("air.2m.gauss.2019.nc")

nc_file
```

Out[34]:

```
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF4_CLASSIC data model, file format HDF5):
    Conventions: COARDS
    title: mean daily NMC reanalysis (2014)
    history: created 2017/12 by Hoop (netCDF2.3)
    description: Data is from NMC initialized reanalysis
(4x/day).  It consists of T62 variables interpolated to
pressure surfaces from model (sigma) surfaces.
    platform: Model
    dataset_title: NCEP-NCAR Reanalysis 1
    References: http://www.psl.noaa.gov/data/gridded/data.ncep.reanalysis.html
    dimensions(sizes): lat(94), lon(192), time(365), nbnds(2)
    variables(dimensions): float32 lat(lat), float32 lon(lon), float64 time(time),
float32 air(time,lat,lon), float64 time_bnds(time,nbnds)
    groups:
```

In [35]:

```
nc_file.variables
```

Out[35]:

```
{'lat': <class 'netCDF4._netCDF4.Variable'>
 float32 lat(lat)
     units: degrees_north
     actual_range: [ 88.542 -88.542]
     long_name: Latitude
     standard_name: latitude
     axis: Y
 unlimited dimensions:
 current shape = (94,)
 filling on, default _FillValue of 9.969209968386869e+36 used,
 'lon': <class 'netCDF4._netCDF4.Variable'>
 float32 lon(lon)
     units: degrees_east
     long_name: Longitude
     actual_range: [  0.    358.125]
     standard_name: longitude
     axis: X
 unlimited dimensions:
 current shape = (192,)
 filling on, default _FillValue of 9.969209968386869e+36 used,
 'time': <class 'netCDF4._netCDF4.Variable'>
 float64 time(time)
     long_name: Time
     delta_t: 0000-00-01 00:00:00
     avg_period: 0000-00-01 00:00:00
     standard_name: time
     axis: T
     units: hours since 1800-01-01 00:00:0.0
     coordinate_defines: start
     actual_range: [1919712. 1928448.]
 unlimited dimensions: time
 current shape = (365,)
 filling on, default _FillValue of 9.969209968386869e+36 used,
 'air': <class 'netCDF4._netCDF4.Variable'>
 float32 air(time, lat, lon)
     long_name: mean Daily Air temperature at 2 m
     units: degK
     precision: 2
     least_significant_digit: 1
     GRIB_id: 11
     GRIB_name: TMP
     var_desc: Air temperature
     dataset: NCEP Reanalysis Daily Averages
     level_desc: 2 m
     statistic: Mean
     parent_stat: Individual Obs
     missing_value: -9.96921e+36
     valid_range: [150. 400.]
     actual_range: [171.8   316.625]
 unlimited dimensions: time
 current shape = (365, 94, 192)
 filling on, default _FillValue of 9.969209968386869e+36 used,
 'time_bnds': <class 'netCDF4._netCDF4.Variable'>
 float64 time_bnds(time, nbnds)
 unlimited dimensions: time
 current shape = (365, 2)
 filling on, default _FillValue of 9.969209968386869e+36 used}
```

In [36]:

```
tas = nc_file.variables["air"]

tas
```

Out[36]:

```
<class 'netCDF4._netCDF4.Variable'>
float32 air(time, lat, lon)
    long_name: mean Daily Air temperature at 2 m
    units: degK
    precision: 2
    least_significant_digit: 1
    GRIB_id: 11
    GRIB_name: TMP
    var_desc: Air temperature
    dataset: NCEP Reanalysis Daily Averages
    level_desc: 2 m
    statistic: Mean
    parent_stat: Individual Obs
    missing_value: -9.96921e+36
    valid_range: [150. 400.]
    actual_range: [171.8   316.625]
unlimited dimensions: time
current shape = (365, 94, 192)
filling on, default _FillValue of 9.969209968386869e+36 used
```

In [37]:

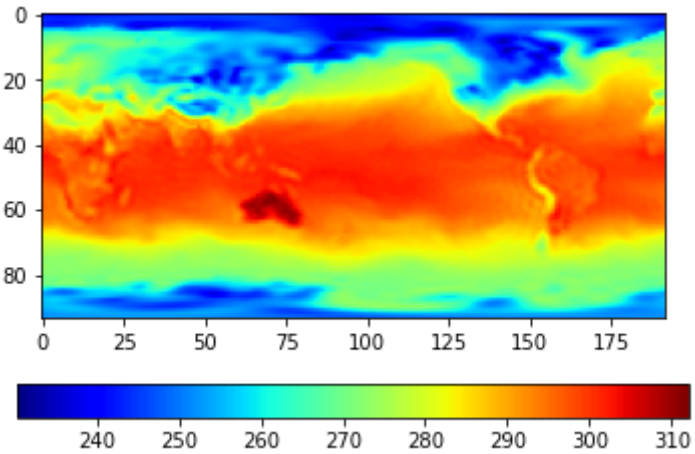
```
tas.shape # time, lat, lon
```

Out[37]:

```
(365, 94, 192)
```

In [38]:

```
plt.imshow(tas[0], cmap = "jet") # 간단하게 자료 확인
plt.colorbar(orientation = "horizontal")

plt.show()
```



## (예제) NetCDF4 파일 저장

In [39]:

```
fw = Dataset("new_netcdf.nc", "w")

fw.createDimension("TIME", 365)
fw.createDimension("LATITUDE", 94)
fw.createDimension("LONGITUDE", 192)

time = fw.createVariable("TIME", "f", ("TIME",)) # "f": float
time[:] = nc_file.variables["time"][:]   # 앞서 불러온 nc파일에서 time 정보를 읽어서, 새로 만든 nc파일에 쓰기
time.units = "hours since 1800-01-01 00:00:0.0"

lat = fw.createVariable("LATITUDE", "f", ("LATITUDE",))
lat[:] = nc_file.variables["lat"][:]

lon = fw.createVariable("LONGITUDE", "f", ("LONGITUDE",))
lon[:] = nc_file.variables["lon"][:]

new_tas = fw.createVariable("TAS","f", ("TIME", "LATITUDE", "LONGITUDE"))
new_tas[:] = tas[:]
new_tas.missing_value = -9999.

fw.close()
```

In [40]:

```
nc_new = Dataset("new_netcdf.nc")

nc_new.variables
```

Out[40]:

```
{'TIME': <class 'netCDF4._netCDF4.Variable'>
 float32 TIME(TIME)
     units: hours since 1800-01-01 00:00:0.0
 unlimited dimensions:
 current shape = (365,)
 filling on, default _FillValue of 9.969209968386869e+36 used,
 'LATITUDE': <class 'netCDF4._netCDF4.Variable'>
 float32 LATITUDE(LATITUDE)
 unlimited dimensions:
 current shape = (94,)
 filling on, default _FillValue of 9.969209968386869e+36 used,
 'LONGITUDE': <class 'netCDF4._netCDF4.Variable'>
 float32 LONGITUDE(LONGITUDE)
 unlimited dimensions:
 current shape = (192,)
 filling on, default _FillValue of 9.969209968386869e+36 used,
 'TAS': <class 'netCDF4._netCDF4.Variable'>
 float32 TAS(TIME, LATITUDE, LONGITUDE)
     missing_value: -9999.0
 unlimited dimensions:
 current shape = (365, 94, 192)
 filling on, default _FillValue of 9.969209968386869e+36 used}
```

# 2-D. HDF5 (.hdf5)

## (1) HDF5 파일 읽기

| | |
|---|---|
| **import h5py** | • HDF5 패키지 |
| **hdf = h5py.File(**파일 경로, "r"**)** | • HDF5 파일 읽기 |
| **keys = []**<br>**hdf.visit(keys.append)** | • HDF5 파일에 포함된 모든 키 확인 |
| **hdf[키]** | • HDF5 파일에서 특정 변수만 불러오기 |
| **for attr, val in hdf[키].attrs.items():**<br>  **print(**attr, val**)** | • 특정 변수의 속성 확인 |
| **hdf[키][:]** | • 특정 변수를 Numpy ndarray 배열로 저장 |
| **hdf.close()** | • 파일 닫기 |

## (2) HDF5 파일 저장

| | |
|---|---|
| **hdf = h5py.File(**파일 경로, "w"**)** | • HDF5 파일 생성 |
| **dataset = hdf.create_dataset(**키, data = array**)** | • 생성한 HDF5 파일에서 키와 값 입력 |
| **dataset.dims[#].label** = 차원 이름 | • 해당 키의 #-차원 설정 |
| **dataset.attrs[**속성 이름**]** = 속성 내용 | • 해당 키의 속성 설정 |
| **hdf.close()** | • 파일 닫고 저장 |

## (예제) HDF5 파일 읽기

In [41]:

```
import h5py

hdf = h5py.File("3B-HHR.MS.MRG.3IMERG.20180101-S000000-E002959.0000.V06B.HDF5", "r")

print(type(hdf))
```

```
<class 'h5py._hl.files.File'>
```

In [42]:

```
hdf_keys = []
hdf.visit(hdf_keys.append)

for i, key in enumerate(hdf_keys):
    print(i, key)
```

```
0 Grid
1 Grid/HQobservationTime
2 Grid/HQprecipSource
3 Grid/HQprecipitation
4 Grid/IRkalmanFilterWeight
5 Grid/IRprecipitation
6 Grid/lat
7 Grid/lat_bnds
8 Grid/latv
9 Grid/lon
10 Grid/lon_bnds
11 Grid/lonv
12 Grid/nv
13 Grid/precipitationCal
14 Grid/precipitationQualityIndex
15 Grid/precipitationUncal
16 Grid/probabilityLiquidPrecipitation
17 Grid/randomError
18 Grid/time
19 Grid/time_bnds
```

In [43]:

```
idx = 13

print(hdf_keys[idx])
```

```
Grid/precipitationCal
```

In [44]:

```
pr = hdf[hdf_keys[idx]]

for name, val in pr.attrs.items():
    print(f"Name: {name}₩nValue: {val}₩n")
```

```
Name: DimensionNames
Value: b'time,lon,lat'

Name: Units
Value: b'mm/hr'

Name: units
Value: b'mm/hr'

Name: coordinates
Value: b'time lon lat'

Name: _FillValue
Value: -9999.900390625

Name: CodeMissingValue
Value: b'-9999.9'

Name: DIMENSION_LIST
Value: [array([<HDF5 object reference>], dtype=object)
 array([<HDF5 object reference>], dtype=object)
 array([<HDF5 object reference>], dtype=object)]
```

In [45]:

```
pr_array = pr[:]

print(pr_array.shape)
```

```
(1, 3600, 1800)
```

In [46]:

```
lons = hdf[hdf_keys[9]][:]
lats = hdf[hdf_keys[6]][:]

print(lons.shape, lats.shape, pr.shape)
```

```
(3600,) (1800,) (1, 3600, 1800)
```

In [47]:

```
print(lons[[0, -1]], lats[[0, -1]], pr_array.min(), pr_array.max())
```

```
[-179.95  179.95] [-89.95  89.95] -9999.9 73.439995
```

In [48]:

```
hdf.close()

pr_array = pr_array[0].T

print(pr_array.shape)
```

(1800, 3600)

In [49]:

```
total_size = pr_array.shape[0] * pr_array.shape[1]
missing = (pr_array == -9999.9)

print(f"Missings = {missing.sum() / total_size * 100 :.1f}%")
```

Missings = 26.8%

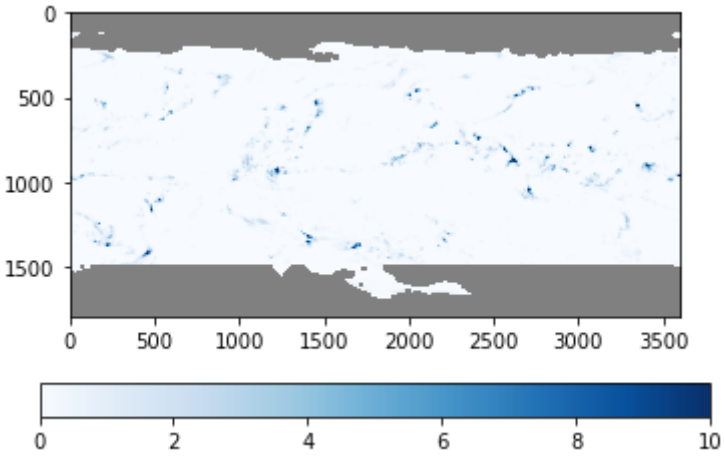
In [50]:

```
zero = (pr_array == 0)

print(f"No-rain = {zero.sum() / total_size * 100 :.1f}%")
```

No-rain = 62.9%

In [51]:

```
pr_array_masked = np.ma.masked_equal(pr_array, -9999.9)

cm = plt.cm.get_cmap("Blues")
cm.set_bad(color = "grey")

plt.imshow(pr_array_masked, cmap = cm, vmin = 0, vmax = 10)
plt.colorbar(orientation = "horizontal")

plt.show()
```




## (예제) HDF5 파일 저장

In [52]:

```
with h5py.File("new_hdf.hdf5", "w") as f:
    lon_dset = f.create_dataset("lon", data = lons)
    lat_dset = f.create_dataset("lat", data = lats)
    pr_dset = f.create_dataset("pre", data = pr_array)

    lon_dset.dims[0].label = "lon"
    lat_dset.dims[0].label = "lat"

    pr_dset.dims[0].label = "lat"
    pr_dset.dims[1].label = "lon"

    pr_dset.attrs["_FillValue"] = -9999.9
    pr_dset.attrs["units"] = "mm/hr"
    pr_dset.attrs["coordinates"] = "lat lon"
```