





- ▼ 자료형 np.ndarray를 생성하는 함수 np.array()
- ▼ 돗수분포표 hist()에서 매개변수 bins
- 난수 발생 np.random.choice() np.random.rand(), np.random.randint()
- ★ 논리 mask 생성과 활용



- ☑ 함수 np.array()로 다양한 차수의 자료형 np.ndarray를 생성할 수 있다.
- ☑ 함수 plt.hist()에서 매개변수 bins를 구간으로 지정하는 의미를 알고 활용할 수 있다.
- ☑ 지정한 수 중에서 난수를 선택하는 np.random.choice()를 활용할 수 있다.
- 🗹 난수 발생 np.random.rand()와 np.random.randint()를 활용할 수 있다.

LESSON 01

numpy array 생성 및 활용



[기 numpy array 생성 및 활용



→ matplotlib.pyplot.hist



matplotlib.pyplot.hist(x, bins=None, range=None, density=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)

🌃 numpy array 생성 및 활용



⊸ hist()

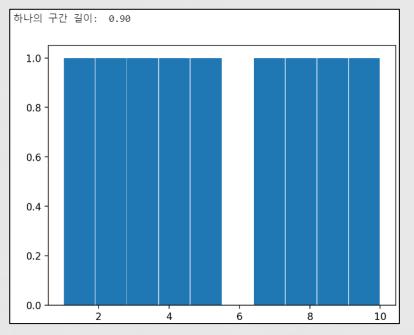


🤪 2번째 매개변수

- ☑ bins int 또는 시퀀스 또는 str, 기본값: (기본값: 10)
 - ★ rcParams["hist.bins"]=2 로 실행시간에 조정 가능
- ☑ bins가 정수인 경우 범위 내 동일한 너비의 bin 수를 정의
- ☑ bins가 시퀀스인 경우 첫 번째 bin의 왼쪽 가장자리와 마지막 bin의 오른쪽 가장자리를 포함하여 bin 구간의 시퀀스를 정의
 - 저장소의 간격이 동일하지 않을 수 있으며, 마지막(가장 오른쪽) 상자를 제외한 모든 상자는 반쯤 열려 있음(불포함)
- ✓ 즉, bins 가 다음과 같은 경우 : [1, 2, 3, 4]
 - → 첫 번째 bin은 (1을 포함하지만 2는 제외)이고 두 번째 bin부터는 왼쪽 이상, 오른쪽 미만
 - ◆ 그러나 마지막 bin은 4를 포함
 - 즉, 각 구간은 [1, 2), [2, 3), [3, 4]를 의미



- → hist 속성 rwidth
- 기본이 1로 가로 폭이 100%
- 기본이 bins는 10



```
import matplotlib.pyplot as plt

data = [1, 2, 3, 4, 5, 7, 8, 9, 10]

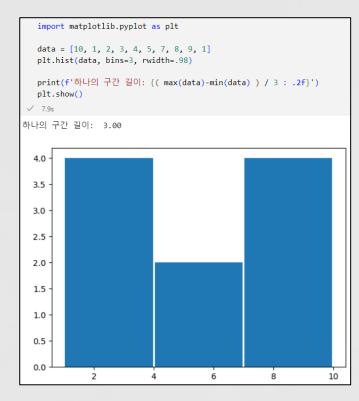
plt.hist(data, rwidth=.98) # 기본 bins=10

print(f'하나의 구간 길이: {( max(data)-min(data) ) / 10 : .2f}')
plt.show()
```



→ hist 속성 bins 수정







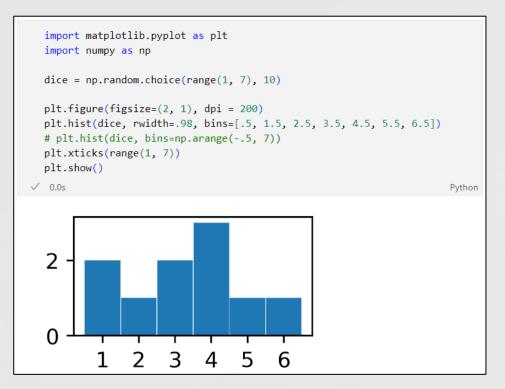
→ hist 속성 bins 수정



```
import matplotlib.pyplot as plt
   data = [10, 1, 2, 3, 4, 5, 7, 8, 9, 1]
   plt.rcParams["hist.bins"] = 2
   plt.hist(data, rwidth=.98)
   print(f'하나의 구간 길이: {( max(data)-min(data) ) / 2 : .2f}')
   plt.show()
✓ 0.1s
하나의 구간 길이: 4.50
  3 -
```



→ numpy 라이브러리를 활용하여 작성한 코드



numpy 라이브러리를 활용할 경우 장점!

- ① 코드가 간결해짐
- ② 코드 실행 속도가 빠름



→ 1부터 6까지 숫자를 랜덤으로 추출한 결과를 히스토그램으로 시각화하기

```
import matplotlib.pyplot as plt
  import numpy as np
  dice = np.random.choice(range(1, 7), 1000000,
                         p = [0.15, 0.25, 0.3, 0.1, 0.1, 0.1]
  plt.figure(figsize=(3, 2), dpi = 150)
  plt.hist(dice, rwidth=.98, bins=np.arange(.5, 7))
  # plt.hist(dice, rwidth=.98, bins=[.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5])
  plt.xticks(range(1, 7))
  plt.show()

√ 0.1s

 300000
 200000
 100000
                                                   6
```

추출 시행 횟수를 늘리니, p 속성에 설정했던 확률 값에 따라 각 숫자가 추출됨을 확인 할 수 있습니다. (큰 수의 법칙)



→ numpy 라이브러리를 활용하여 작성한 코드

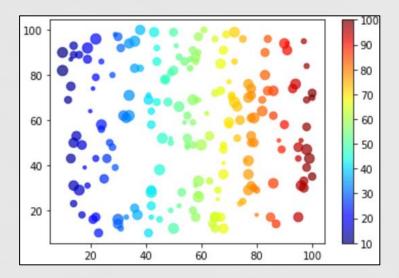
```
import matplotlib.pyplot as plt import numpy as np

10이상 100미만의 정수 중 200개를 뽑는다.

x = np.random.randint(10, 100, 200)
y = np.random.randint(10, 100, 200)
size = np.random.rand(200) * 100

Day 1사이의 실수 200개를 뽑는다.

plt.scatter(x, y, s = size, c = x, cmap = 'jet', alpha = 0.7)
plt.colorbar()
plt.show()
```



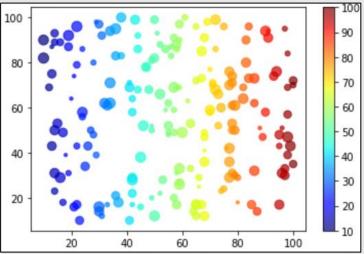


→ 파이썬 리스트를 활용하여 작성한 코드

```
import matplotlib.pyplot as plt
import random

x = []
y = []
size = []

for i in range(200):
    x.append(random.randint(10, 100))
    y.append(random.randint(10, 100))
    size.append(random.randint(10, 100))
    plt.scatter(x, y, s = size, c = x, cmap = 'jet', alpha = 0.7)
plt.colorbar()
plt.show()
```



III numpy array 생성 및 활용



→ 리스트로 ndarray(N-Dimensional Array) 만들기

```
import numpy as np
a = np.array([1, 2, 3, 4])
print(a)
[1 2 3 4]
```

[11] numpy array 생성 및 활용



→ numpy array의 인덱싱(Indexing), 슬라이싱(Slicing)

```
import numpy as np
a = np.array([1, 2, 3, 4])
print(a[1], a[-1])
print(a[1:])
2 4
[2 3 4]
```



→ zeros(), ones(), eye() 함수로 numpy array 만들기

```
import numpy as np
a = np.zeros(10)
print(a)
b = np.ones(10)
print(b)

c = np.eye(3) 3행 × 3열의 단위 행렬 생성

[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
[[1. 0. 0.]
[[0. 1. 0.]
[[0. 0. 1.]]
```

III numpy array 생성 및 활용



→ 연속된 숫자의 numpy array 만들기

```
import numpy as np
print(np.arange(3))
print(np.arange(3, 7))
print(np.arange(3, 7, 2))

[0 1 2]
[3 4 5 6]
[3 5]
```



→ 연속된 실수의 numpy array 만들기

```
import numpy as np

a = np.arange(1, 2, 0.1) # 1이상 2미만 구간에서 0.1 간격으로 실수 생성
b = np.linspace(1, 2, 11) #1부터 2까지 10개 구간으로 나눈 실수 생성
print(a)
print(b)

Python

[1. 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9]
[1. 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```



→ 연속된 실수의 numpy array 만들기

```
import numpy as np
a = np.arange(-np.pi, np.pi, np.pi/10)
b = np.linspace(-np.pi, np.pi, 20)
print(a)
print(b)
[-3.14159265 -2.82743339 -2.51327412 -2.19911486 -1.88495559 -1.57079633
-1.25663706 -0.9424778 -0.62831853 -0.31415927 0.
                                                            0.31415927
 0.62831853 0.9424778
                        1.25663706 1.57079633 1.88495559 2.19911486
  2.51327412 2.827433391
[-3.14159265 -2.81089869 -2.48020473 -2.14951076 -1.8188168 -1.48812284
 -1.15742887 -0.82673491 -0.49604095 -0.16534698 0.16534698 0.49604095
 0.82673491 1.15742887 1.48812284 1.8188168
                                                2.14951076 2.48020473
  2.81089869 3.14159265]
```

In numpy array 생성 및 활용



→ numpy array에 값을 한꺼번에 더하기

```
import numpy as np
a = np.zeros(10) + 5
print(a)
[5. 5. 5. 5. 5. 5. 5. 5. 5.]
```

III numpy array 생성 및 활용



→ numpy array에 함수 적용하기



→ numpy를 활용한 그래프 그리기

```
import numpy as np
import matplotlib.pyplot as plt
a = np.arange(-np.pi, np.pi, np.pi/100)
plt.plot(a, np.sin(a))
plt.show()
  1.00
  0.75
  0.50
  0.25
  0.00
 -0.25
 -0.50
 -0.75
 -1.00
         -3
                -2
                              Ó
```



→ numpy를 활용한 다양한 그래프 그리기

```
import numpy as np
import matplotlib.pyplot as plt
a = np.arange(-np.pi, np.pi, np.pi/100)
plt.plot(a, np.sin(a), label='np.sin(a)')
plt.plot(a, np.cos(a), label='np.cos(a)')
plt.plot(a + np.pi / 2, np.sin(a), label='... np.sin(a)')
plt.legend()
                    1.00 -
                           np.sin(a)
plt.show()
                           np.cos(a)
                           ... np.sin(a)
                    0.75
                    0.50
                    0.25
                    0.00 -
                   -0.25
                   -0.50
                   -0.75
                   -1.00
                             -2
```



→ Numpy 활용의 장점

❷ numpy 라이브러리를 활용하여 작성한 코드

```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0., 5., 0.2)

plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

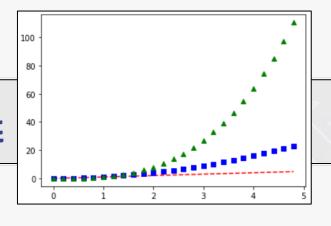
❷ 파이썬 리스트를 활용하여 작성한 코드

```
import matplotlib.pyplot as plt

a = []
b = []
c = []

for i in range(0, 50, 2):
    a.append(i / 10)
    b.append(i / 10) ** 2)
    c.append(i / 10) ** 3)

plt.plot(a, a, 'r--', a, b, 'bs', a, c, 'g^')
plt.show()
```



numpy 라이브러리를 활용할 경우 보다 적은 수의 코드로 그리고 간결하게 원하는 결과를 얻을 수 있습니다. LESSON 02

numpy 논리 mask 활용



☑근 numpy 논리 mask 활용



⊸ 마스크(Mask) 만들고 적용하기

```
import numpy as np

a = np.arange(-5, 5) # 데이터 만들기
print(a)

print(a<0) # 마스크 (Mask) 만들기
print(a[a<0]) # 마스크 적용하기

[-5 -4 -3 -2 -1 0 1 2 3 4]
[ True True True True True False False False False False False [-5 -4 -3 -2 -1]
```



→ 마스크(Mask) 연결해서 사용하기

```
import numpy as np
     a = np.arange(-5, 5)
     print(a)
  [-5 -4 -3 -2 -1 0 1 2 3 4]
     mask1 = abs(a) > 3
     print(mask1)
     print(a[mask1])
  [ True True False False False False False False True]
  [-5 -4 4]
     mask2 = (abs(a) \% 2) == 0
     print(mask2)
     print(a[mask2])
[44] V 0.0s
  [False True False True False True False True]
  [-4 -2 0 2 4]
```



→ 마스크(Mask) 연결해서 사용하기

```
print(mask1)
     print(mask2)
     print(mask1 + mask2)
     print(mask1 * mask2)
[47] 		 0.0s
  [ True True False False False False False False True]
  [False True False True False True False True]
  [ True True False True False True False True]
  [False True False False False False False False False True]
     print(a[mask1 + mask2]) # 둘 중 하나의 조건이라도 참일 경우
     print(a[mask1 * mask2]) # 두 가지 조건이 모두 참일 경우
  [-5 -4 -2 0 2 4]
  [-4 4]
```



→ numpy 라이브러리를 사용하여 재미있는 산점도 표현

```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.randint(-100, 100, 1000)
                                                          100
y = np.random.randint(-100, 100, 1000)
                                                                                                                 - 75
                                                           75
mask1 = (abs(x) > 50)
                                                                                                                 - 50
                                                           50
mask2 = (abs(y) > 50)
x = x[mask1 + mask2] # 둘 중 하나라도 참이면 포함
                                                                                                                 25
y = y[mask1 + mask2]
                                                                                                                 - 0
plt.scatter(x, y, c=x, cmap='jet', alpha=0.7)
plt.colorbar()
                                                          -25
                                                                                                                 - -25
plt.show()
                                                          -50
                                                                                                                 --50
                                                          -75
                                                                                                                 -75
```

-100

-100

-50

0

50

100

-100

SUMMARY

학습정긴





...

- Matplotlib hist()
 - matplotlib.pyplot.hist(x, bins=None, …)
- Numpy 난수 발생의 장점
 - 여러 배열 생성 가능
 - np.random.rand(d0, d1, d2, ···)
 - np.random.randint(low, high=None, size=None, dtype=int)
- ⊙ 주사위의 시행 횟수를 늘리면 결국 1/6 확률
 - random.choice(a, size=None, replace=True, p=None)

```
dice = np.random.choice(range(1, 7), 1000000,

p = [0.15, 0.25, 0.3, 0.1, 0.1, 0.1])
```

- 👸 마스크 생성과 활용
 - >> Ndarray에 마스크를 적용해서 조건에 맞는 것을 선택 가능



