NumPy array

Comparisons, Masks, and Boolean Logic

NumPy array မှ ဒေတာများ ထုတ်ယူခြင်း (extraction)၊ ရေတွက်ခြင်း (counting)၊ လိုအပ်သလို
ပြောင်းလဲခြင်း (modification)ပြုလုပ်ပုံများကို ဖော်ပြထားသည်။

ဥပမာ (Example: Counting Rainy Days)

၂၀၁၄ ခုနှစ်တွင် Seattle မြို့၌ ရွာသွန်းသည့် နေ့စဉ်မိုးရေချိန် (daily rainfall statistics)ကို မီလီ လီတာဖြင့် ဖော်ထားသည်။ Precipitation = မိုးရွာသည့် မိုးရေချိန်လက်မ သို့မဟုတ် ဆီးနှင်းကျသည့် ပမာဏ

numpy နှင့် pandas ကို import လုပ်သည်။

In [1]:
```
import numpy as np
import pandas as pd
```

နေ့စဉ်မိုးရေချိန် (daily rainfall statistics)ကို ဖတ်သည်။ မီလီ လီတာ ကို လက်မသို့ ပြောင်းသည်။

inches.shape ဖြင့် ဒေတာ၏ row မည်မျှ ရှိသည်ကို စစ်ဆေးသည်။

In [2]:
```
# use pandas to extract rainfall inches as a NumPy array
rainfall = pd.read_csv('data/Seattle2014.csv')['PRCP'].values
inches = rainfall / 254.0  # 1/10mm -> inches
inches.shape
```

Out[2]:
```
(365,)
```

Array မှ ဒေတာများကို ကြည့်သည်။

In [3]:
```
inches
```

Out[3]:
```
array([0.        , 0.16141732, 0.05905512, 0.        , 0.        ,
       0.01181102, 0.48031496, 0.38188976, 0.22834646, 0.16929134,
       0.83858268, 0.05905512, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.01968504, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.3503937 , 0.8503937 , 0.        ,
       0.09055118, 0.07874016, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.2007874 , 0.01968504,
       0.72047244, 0.66929134, 0.18110236, 0.07086614, 0.37007874,
       0.46062992, 1.03937008, 0.57086614, 0.5984252 , 0.03937008,
       0.11811024, 0.11023622, 0.0984252 , 0.24015748, 0.51181102,
       0.01181102, 0.        , 0.        , 0.        , 0.01968504,
       0.7519685 , 0.42125984, 0.6496063 , 1.83858268, 0.11811024,
       0.        , 1.27165354, 0.16929134, 0.74015748, 0.        ,
       0.        , 0.01968504, 0.27165354, 0.31889764, 1.09055118,
       0.01181102, 0.        , 0.01968504, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.16141732, 0.14173228,
       0.01181102, 0.87007874, 0.5511811 , 0.        , 0.        ,
       0.        , 0.        , 0.0984252 , 0.        , 0.18110236,
       0.        , 0.        , 0.18110236, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.01968504,
       0.42913386, 0.72834646, 0.        , 0.53937008, 0.        ,
       0.2007874 , 0.55905512, 0.3503937 , 0.48818898, 0.        ,
       0.12992126, 0.27165354, 0.        , 0.        , 0.        ,
       0.        , 0.        , 1.31102362, 0.62992126, 0.2007874 ,
       0.        , 0.        , 0.53937008, 0.07874016, 0.01968504,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.1496063 , 0.        , 0.22047244,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.07086614, 0.2519685 , 0.        ,
       0.01968504, 0.14173228, 0.0511811 , 0.        , 0.03149606,
       0.01181102, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.07086614, 0.09055118, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.01181102, 0.75984252, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.01968504, 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.01968504, 0.5       , 0.8503937 ,
       0.        , 0.03937008, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.33070866, 0.0511811 , 0.        , 0.11811024,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.01968504,
       0.01181102, 0.        , 0.        , 0.        , 0.01181102,
       0.72047244, 0.7992126 , 0.16929134, 0.3503937 , 0.        ,
       0.        , 0.03149606, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.01181102, 0.29133858, 0.        ,
       0.2992126 , 0.27952756, 0.33858268, 0.        , 0.12992126,
       0.59055118, 0.        , 0.46062992, 0.03937008, 1.25984252,
       0.37007874, 0.16141732, 0.24015748, 0.05905512, 0.03149606,
       0.5       , 0.01968504, 1.        , 0.66929134, 0.        ,
       0.07086614, 0.42913386, 0.16141732, 0.18897638, 0.16141732,
       0.        , 0.        , 0.2007874 , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.14173228, 0.5984252 ,
       0.01968504, 0.46850394, 0.0511811 , 0.72047244, 0.01181102,
       0.12992126, 1.3503937 , 0.14173228, 0.        , 0.        ,
       0.        , 0.        , 0.03149606, 0.11811024, 0.29133858,
       0.        , 0.35826772, 0.38976378, 0.51181102, 0.27165354,
```

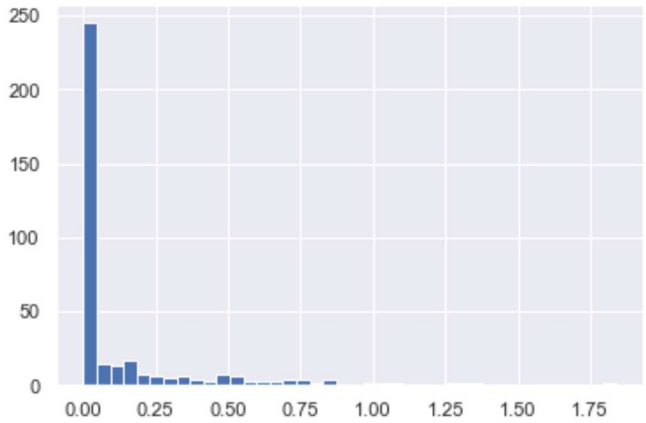
January 1 မှ December 31, 2014 အထိ ၃၆၅ ရက်ဖြစ်သောကြောင့် ၃၆၅ row ရှိသည်။ The array contains 365 values, giving daily rainfall in inches from January 1 to December 31, 2014.

Data visualization လုပ်ရန် histogram of rainy days ဂရပ်ဆွဲကြည့်သည်။ matplotlib.pyplot နှင့် seaborn ကို import လုပ်သည်။

In [4]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn; seaborn.set()  # set plot styles
```

plt.hist() ဖြင့် histogram ဂရပ်ဆွဲသည်။

In [5]:
```
plt.hist(inches, 40);
```



Comparison Operators as ufuncs

np.array() ဖြင့် [1, 2, 3, 4, 5] ဒေတာများထည့်ပြီး NumPy array တစ်ခုတည်ဆောက်သည်။

In [6]:
```
x = np.array([1, 2, 3, 4, 5])
```

NumPy array ဖြစ်သည့် x ထဲမှာ ၃ ထက် ငယ်သည် ကိန်းများကို ဖော်ပြခိုင်းသည်။ ၃ ထက် ငယ်လျှင် True ဖြစ်ပြီး ကြီးလျှင် False ဖြစ်သည်။

In [7]:
```
x < 3  # less than
```

Out[7]:
```
array([ True,  True, False, False, False])
```

NumPy array ဖြစ်သည့် x ထဲမှာ ၃ ထက် ကြီးသည် ကိန်းများကို ဖော်ပြခိုင်းသည်။ ၃ ထက် ကြီးလျှင် True ဖြစ်ပြီး ငယ်လျှင် False ဖြစ်သည်။

In [8]:
```
x > 3  # greater than
```

Out[8]:
```
array([False, False, False,  True,  True])
```

၃ နှင့် ညီပြီး ၃ ငယ်လျှင် True ဖြစ်ပြီး ၃ ထက် ကြီးလျှင် False ဖြစ်သည်။

In [9]:
```
x <= 3  # less than or equal
```

Out[9]:
```
array([ True,  True,  True, False, False])
```

၃ နှင့် ညီပြီး ၃ ထက် ကြီးလျှင် True ဖြစ်ပြီး ၃ ငယ်လျှင် False ဖြစ်သည်။

```
In [10]: x >= 3  # greater than or equal
```

```
Out[10]: array([False, False,  True,  True,  True])
```

၃ နှင့် မညီလျှင် True ဖြစ်သည်။

```
In [11]: x != 3  # not equal
```

```
Out[11]: array([ True,  True, False,  True,  True])
```

၃ နှင့် ညီလျှင် True ဖြစ်သည်။

```
In [12]: x == 3  # equal
```

```
Out[12]: array([False, False,  True, False, False])
```

Element-wise comparison of two arrays

NumPy array ဖြစ်သည့် x ကို ၂ နှင့် မြှောက်သည်။ x ကို နှစ်ထပ်ကိန်းတင်သည်။

ဖြစ်ပေါ်လာသည့် ထို array နှစ်ခု ရှိ element များကို အချင်းချင်းကို comparison လုပ်သည်။ တူညီလျှင် True ဖြစ်ပြီး မတူညီလျှင် False ဖြစ်သည်။

```
In [13]: (2 * x) == (x ** 2)
```

```
Out[13]: array([False,  True, False, False, False])
```

As in the case of arithmetic operators, the comparison operators are implemented as ufuncs in NumPy; for example, when you write `x < 3`, internally NumPy uses `np.less(x, 3)`. A summary of the comparison operators and their equivalent ufunc is shown here:

Operator	Equivalent ufunc	Operator	Equivalent ufunc
==	np.equal	!=	np.not_equal
<	np.less	<=	np.less_equal
>	np.greater	>=	np.greater_equal

Just as in the case of arithmetic ufuncs, these will work on arrays of any size and shape. Here is a two-dimensional example:

Random NumPy array တစ်ခုတည်ဆောက်သည်။

```
In [14]: rng = np.random.RandomState(0)
         x = rng.randint(10, size=(3, 4))
         x
```

```
Out[14]: array([[5, 0, 3, 3],
                [7, 9, 3, 5],
                [2, 4, 7, 6]])
```

x ထဲမှ element များ ၆ ထက် ငယ် မငယ် စစ်သည်။

```
In [15]: x < 6
```

```
Out[15]: array([[ True,  True,  True,  True],
                [False, False,  True,  True],
                [ True,  True, False, False]])
```

In each case, the result is a Boolean array, and NumPy provides a number of straightforward patterns for working with these Boolean results.

Working with Boolean Arrays

Given a Boolean array, there are a host of useful operations you can do. We'll work with `x`, the two-dimensional array we created earlier.

```
In [16]: print(x)
```

```
[[5 0 3 3]
 [7 9 3 5]
 [2 4 7 6]]
```

Counting entries

To count the number of `True` entries in a Boolean array, `np.count_nonzero` is useful:

x ထဲမှ element များ ဖြ ထက် ငယ်သည့် element အရေအတွက်ကို ဖော်ပြရန် np.count_nonzero() ကို သုံးသည်။

```
In [17]: # how many values less than 6?
         np.count_nonzero(x < 6)
```

```
Out[17]: 8
```

We see that there are eight array entries that are less than 6. Another way to get at this information is to use `np.sum`; in this case, `False` is interpreted as `0`, and `True` is interpreted as `1`:

x ထဲမှ element များ ဖြ ထက် ငယ်လျှင် `True` ဖြစ်သည်။ `True` ၏ တန်ဖိုးက ၁ အဖြစ် ယူသည်။ np.sum()ဖြင့်လည်း ဖော်ပြနိုင်သည်။

```
In [18]: np.sum(x < 6)
```

```
Out[18]: 8
```

The benefit of `sum()` is that like with other NumPy aggregation functions, this summation can be done along rows or columns as well:

row တစ်ခုစီတွင် ရှိသည့် ဖြ ထက်ငယ်သည် element ကိုလည်း ယခုကဲ့သို့ `np.sum(x < 6, axis=1)` ဖော်ပြနိုင်သည်။ ပထမ row တွင် ဖြ ထက်ငယ်သည့် element ၄ခု ၊ ဒုတိယ row တွင် ၂ခု ၊ တတိယ row တွင် ၂ခု

```
In [19]: # how many values less than 6 in each row?
         np.sum(x < 6, axis=1)
```

```
Out[19]: array([4, 2, 2])
```

This counts the number of values less than 6 in each row of the matrix.

If we're interested in quickly checking whether any or all the values are true, we can use (you guessed it) `np.any` or `np.all`:

၈ ထက်ကြီးသည့် element ရှိ မရှိ `np.any(x > 8)` ဖြင့် စစ်သည်။

```
In [20]: # are there any values greater than 8?
         np.any(x > 8)
```

```
Out[20]: True
```

၀ ထက် ငယ်သည့် element ရှိ မရှိ `np.any(x < 0)` ဖြင့် စစ်သည်။

```
In [21]: # are there any values less than zero?
         np.any(x < 0)
```

```
Out[21]: False
```

element များ အားလုံး ၁၀ ထက် ငယ် မငယ် စစ်သည်။

```
In [22]: # are all values less than 10?
         np.all(x < 10)
```

```
Out[22]: True
```

element များ အားလုံး ၆ နှင့် ညီ မညီ စစ်သည်။

```
In [23]: # are all values equal to 6?
         np.all(x == 6)
```

```
Out[23]: False
```

`np.all` and `np.any` can be used along particular axes as well. For example:

`np.all` and `np.any` တို့ ဖြင့် row အလိုက် column အလိုက် စစ်နိုင်သည်။ တစ် row လုံးတွင် ရှိသည့် element အားလုံး ၈ ထက် ငယ် မငယ် စစ်သည်။

```
In [24]: # are all values in each row less than 8?
         np.all(x < 8, axis=1)
```

```
Out[24]: array([ True, False,  True])
```

Boolean operators

Python's *bitwise logic operators*, `&`, `|`, `^`, and `~`.

Like with the standard arithmetic operators, NumPy overloads these as ufuncs which work element-wise on (usually Boolean) arrays.

အောက်တွင် ဥပမာ ဖြင့် ဖော်ပြထားသည်။ အထက်တွင် မိုးရေချိန်လက်မ inches array ကို ရှင်းပြခဲ့ပြီး ဖြစ်သည်။

မိုးရေချိန် လက်မ 0.5 များပြီး `(inches > 0.5)` 1.0 ထက်နည်းသည့် `(inches < 1)` ရက်များကို ဖော်ပြရန်

In [25]:
```
np.sum((inches > 0.5) & (inches < 1))
```

Out[25]: 29

မိုးရေချိန် လက်မ ဝက်မှ တစ်လက်မ အတွင်း မိုးရွာသည့် ရက်ပေါင်းမှာ ၂၉ ရက် ဖြစ်သည်။ So we see that there are 29 days with rainfall between 0.5 and 1.0 inches.

Note that the parentheses here are important–because of operator precedence rules, with parentheses removed this expression would be evaluated as follows, which results in an error:

```
inches > (0.5 & inches) < 1
```

Using the equivalence of *A AND B* and *NOT (NOT A OR NOT B)* (which you may remember if you've taken an introductory logic course), we can compute the same result in a different manner:

In [26]:
```
np.sum(~( (inches <= 0.5) | (inches >= 1) ))
```

Out[26]: 29

Combining comparison operators and Boolean operators on arrays can lead to a wide range of efficient logical operations.

The following table summarizes the bitwise Boolean operators and their equivalent ufuncs:

Operator	Equivalent ufunc	Operator	Equivalent ufunc
`&`	`np.bitwise_and`	`|`	`np.bitwise_or`
`^`	`np.bitwise_xor`	`~`	`np.bitwise_not`

Using these tools, we might start to answer the types of questions we have about our weather data. Here are some examples of results we can compute when combining masking with aggregations:

မိုး မရွာသည့် ရက်ပေါင်း: `print("Number days without rain:      ", np.sum(inches == 0))`

မိုး ရွာသည့် ရက်ပေါင်း: `print("Number days with rain:         ", np.sum(inches != 0))`

မိုးရေချိန် လက်မ ဝက်ထက် ပိုရွာသည့် ရက်ပေါင်း: `print("Days with more than 0.5 inches:", np.sum(inches > 0.5))`

မိုးရေချိန် 0.2 လက်မ ထက်နည်းသည့် မိုးရွာသည့် ရက်ပေါင်း: `print("Rainy days with < 0.2 inches  :", np.sum((inches > 0) & (inches < 0.2)))`

In [27]:
```
print("Number days without rain:      ", np.sum(inches == 0))
print("Number days with rain:         ", np.sum(inches != 0))
print("Days with more than 0.5 inches:", np.sum(inches > 0.5))
print("Rainy days with < 0.2 inches  :", np.sum((inches > 0) &
                                                (inches < 0.2)))
```

```
Number days without rain:       215
Number days with rain:          150
Days with more than 0.5 inches: 37
Rainy days with < 0.2 inches  : 75
```

Boolean Arrays as Masks

```
In [28]: x
```

```
Out[28]: array([[5, 0, 3, 3],
                [7, 9, 3, 5],
                [2, 4, 7, 6]])
```

အခြေအနေတစ်ခုခု ပေး၍ array မှ Boolean array အဖြစ် ပြောင်းနိုင်သည်။ သို့မဟုတ်Boolean array တစ်ခု ပြုလုပ်နိုင်သည်။

We can obtain a Boolean array for this condition easily

```
In [29]: x < 5
```

```
Out[29]: array([[False,  True,  True,  True],
                [False, False,  True, False],
                [ True,  True, False, False]])
```

masking operation:

```
In [30]: x[x < 5]
```

```
Out[30]: array([0, 3, 3, 3, 2, 4])
```

Precipitation = မိုးရွာသည့် မိုးရေချိန်လက်မ သို့မဟုတ် ဆီးနှင်းကျသည့် ပမာဏ

၂၀၁၄ ခုနှစ် မိုးရွာသည့် နေများတွင် (မိုးရာသီ) ပျှမ်းမျှ မိုးရေချိန်လက်မ(Precipitation) `print("Median precip on rainy days in 2014 (inches):  ",`
`np.median(inches[rainy]))`

၂၀၁၄ ခုနှစ် နွေရာသီ ပျှမ်းမျှ ဆီးနှင်းကျသည့် ပမာဏ (လက်မ)(Precipitation) `print("Median precip on summer days in 2014 (inches):  ",`
`np.median(inches[summer]))`

၂၀၁၄ ခုနှစ် နွေရာသီ အများဆုံး ဆီးနှင်းကျသည့် ပမာဏ (လက်မ)(Precipitation) `print("Maximum precip on summer days in 2014 (inches): ",`
`np.max(inches[summer]))`

၂၀၁၄ ခုနှစ် နွေရာသီ မဟုတ်သည့်နေ့နှင့် မိုးရွာသည့်နေများတွင် ပျှမ်းမျှ Precipitation(လက်မ) `print("Median precip on non-summer rainy days (inches):",`
`np.median(inches[rainy & ~summer]))`

What is returned is a one-dimensional array filled with all the values that meet this condition; in other words, all the values in positions at which the mask array is `True` .

We are then free to operate on these values as we wish. For example, we can compute some relevant statistics on our Seattle rain data:

In [31]:
```
# construct a mask of all rainy days
rainy = (inches > 0)

# construct a mask of all summer days (June 21st is the 172nd day)
days = np.arange(365)
summer = (days > 172) & (days < 262)

print("Median precip on rainy days in 2014 (inches):   ",
      np.median(inches[rainy]))
print("Median precip on summer days in 2014 (inches):  ",
      np.median(inches[summer]))
print("Maximum precip on summer days in 2014 (inches): ",
      np.max(inches[summer]))
print("Median precip on non-summer rainy days (inches):",
      np.median(inches[rainy & ~summer]))
```

```
Median precip on rainy days in 2014 (inches):    0.19488188976377951
Median precip on summer days in 2014 (inches):   0.0
Maximum precip on summer days in 2014 (inches):  0.8503937007874016
Median precip on non-summer rainy days (inches): 0.20078740157480315
```

By combining Boolean operations, masking operations, and aggregates, we can very quickly answer these sorts of questions for our dataset.

Aside: Using the Keywords and/or Versus the Operators &/|

One common point of confusion is the difference between the keywords `and` and `or` on one hand, and the operators `&` and `|` on the other hand. When would you use one versus the other?

The difference is this: `and` and `or` gauge the truth or falsehood of *entire object*, while `&` and `|` refer to *bits within each object*.

When you use `and` or `or`, it's equivalent to asking Python to treat the object as a single Boolean entity. In Python, all nonzero integers will evaluate as True. Thus:

In [32]:
```
bool(42), bool(0)
```

Out[32]: (True, False)

In [33]:
```
bool(42 and 0)
```

Out[33]: False

In [34]:
```
bool(42 or 0)
```

Out[34]: True

When you use `&` and `|` on integers, the expression operates on the bits of the element, applying the *and* or the *or* to the individual bits making up the number:

In [35]:
```
bin(42)
```

Out[35]: '0b101010'

In [36]:
```
bin(59)
```

Out[36]: '0b111011'

In [37]:
```
bin(42 & 59)
```

Out[37]:
```
'0b101010'
```

In [38]:
```
bin(42 | 59)
```

Out[38]:
```
'0b111011'
```

`1 = True` `0 = False` ဖြစ်သည်။

NumPy တွင် Boolean value များပါဝင်သည့် array တွင် `1 = True` ၊ `0 = False` ဖြစ်သည်။

Notice that the corresponding bits of the binary representation are compared in order to yield the result.

When you have an array of Boolean values in NumPy, this can be thought of as a string of bits where `1 = True` and `0 = False`, and the result of `&` and `|` operates similarly to above:

In [39]:
```
A = np.array([1, 0, 1, 0, 1, 0], dtype=bool)
A
```

Out[39]:
```
array([ True, False,  True, False,  True, False])
```

In [40]:
```
A = np.array([1, 0, 1, 0, 1, 0], dtype=bool)
B = np.array([1, 1, 1, 0, 1, 1], dtype=bool)
A | B
```

Out[40]:
```
array([ True,  True,  True, False,  True,  True])
```

ValueError ပေးလိမ့်မည်။

`or` ဖြင့် truth or falsehood of the entire array object တစ်ခုလုံး၏ truth သို့မဟုတ် false ရှာဖွေပါက ValueError ပေးလိမ့်မည်။ well-defined value မဟုတ်သောကြောင့် ဖြစ်သည်။

Using `or` on these arrays will try to evaluate the truth or falsehood of the entire array object, which is not a well-defined value:

In [41]:
```
A or B
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-41-ea2c97d9d9ee> in <module>
----> 1 A or B

ValueError: The truth value of an array with more than one element is ambiguou
s. Use a.any() or a.all()
```

Similarly, when doing a Boolean expression on a given array, you should use `|` or `&` rather than `or` or `and`:

In [42]:
```
x = np.arange(10)
(x > 4) & (x < 8)
```

Out[42]:
```
array([False, False, False, False, False,  True,  True,  True, False,
       False])
```

Trying to evaluate the truth or falsehood of the entire array will give the same `ValueError` we saw previously:

```
In [43]: (x > 4) and (x < 8)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-43-eecf1fdd5fb4> in <module>
----> 1 (x > 4) and (x < 8)

ValueError: The truth value of an array with more than one element is ambiguou
s. Use a.any() or a.all()
```

So remember this: `and` and `or` perform a single Boolean evaluation on an entire object, while `&` and `|` perform multiple Boolean evaluations on the content (the individual bits or bytes) of an object. For Boolean NumPy arrays, the latter is nearly always the desired operation.