Programming in Python II

Dr Joel Yang
Joel_YANG@np.edu.sg
Senior Lecturer, School of ICT
NgeeAnn Polytechnic
www.np.edu.sg/ict

Wei-Meng Lee
weimenglee@learn2develop.net
Founder, Developer Learning Solutions
www.learn2develop.net

NGEE ANN POLYTECHNIC

Developer Learning Solutions

Scope of this Course

- The course covers the following:

Day 1

- Advanced Python Programming
  - Lambda Functions
  - Object Oriented Programming
  - Iterators and Generators
  - Modules and Packages

Day 2

- Manipulating Data using NumPy and Pandas
  - NumPy
    - Basics of arrays, matrices, etc

Day 3

  - Pandas
    - Series
    - DataFrames

What Will You Achieve?

- By the end of this course, you will be able to:
    - Understand how to write *functional* code using lambda functions
    - Understand the differences between modules and libraries
    - Understand Object Oriented Programming in Python
    - Use NumPy to manipulate your data using arrays
    - Use Pandas to manipulate tabular data
    - Use Github for code sharing

Total Curriculum Hours

- The total curriculum hours for the course is 40 hours, which is divided into the following:
  - **BEFORE** - Pre-Lesson Preparation: 8 Hours (1 day)
  - **DURING** - Face-to-Face Session: 24 hours (3 days)
  - **AFTER** - Post-Lesson Learning: 8 hours (1 day)

What to Expect For This Module

- We are shifting to a higher gear
- The chapters on NumPy and Pandas will lay the foundation for future modules
    - They may not be very exciting
    - But they are absolutely important for moving forward in the field of data science and machine learning

Lambda Functions

Lambda Function

- Lambda function is also known as an *anonymous function*
    - An *anonymous function* is a function without a name
- A lambda function can take on any number of arguments, but can only contain a *single* expression
    - used when you need a function for a short period of time
    - commonly used when you want to pass a function as an argument to higher-order functions, that is, functions that take other functions as their arguments

Uses of Lambda Functions

- Suppose you have a list:

```
lst = [1,2,3,4,5,6,7,8,9]
```

- Can you write a function to do the following:
  - Take in a list and return all the even numbers
  - Take in a list and return all numbers greater than 5

Getting all even numbers

```
def even_nums(l):
    result = []
    for n in l:
        if n % 2 == 0:
            result.append(n)
    return result

print(even_nums(lst))
```

Getting all numbers greater than 5

```
def num_gt_5(l):
    result = []
    for n in l:
        if n > 5:
            result.append(n)
    return result

print(num_gt_5(lst))
```

Comparing the 2 Functions

```
def even_nums(l):
    result = []
    for n in l:
        if n % 2 == 0:
            result.append(n)
    return result

print(even_nums(lst))
```

```
def num_gt_5(l):
    result = []
    for n in l:
        if n > 5:
            result.append(n)
    return result

print(num_gt_5(lst))
```

- From this exercise, you can see that the 2 functions are very similar; except the expression in bold

- It would be more efficient to write a function, say, **filter()**, that performs the above, and you just need to specify the condition

`filter()` Function

- There is indeed a **`filter()`** function!

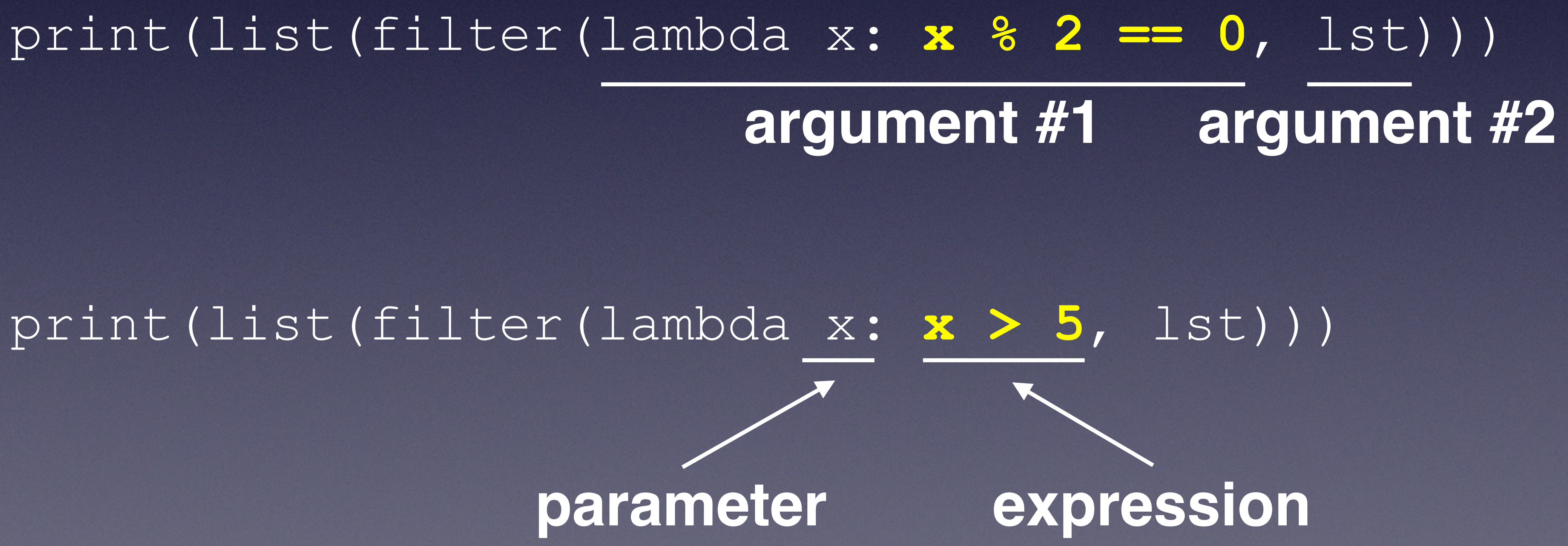
- You can just pass in your condition:

```
lst = [1,2,3,4,5,6,7,8,9]

print(list(filter(lambda x: x % 2 == 0, lst)))

print(list(filter(lambda x: x > 5, lst)))
```

`filter()` Function

- You can just pass in your condition:

```
lst = [1,2,3,4,5,6,7,8,9]
```



```
print(list(filter(lambda x: x % 2 == 0, lst)))
```

argument #1 argument #2

```
print(list(filter(lambda x: x > 5, lst)))
```

parameter expression

Writing a function that uses Lambda functions

function

```
lf = lambda x : x % 2 == 0
```

```
def my_filter(lf, lst):
    result = []
    for n in lst:
        if (lf(n)):
            result.append(n)
    return result

print(my_filter(lambda x: x % 2 == 0, lst))
print(my_filter(lambda x: x > 5, lst))
```

Writing a function that uses Lambda functions

```
def filter(lf, lst):
    result = []
    for n in lst:
        if (n % 2 == 0):
            result.append(n)
    return result

print(filter(lambda x: x % 2 == 0, lst))
print(filter(lambda x: x > 5, lst))
```

Writing a function that uses Lambda functions

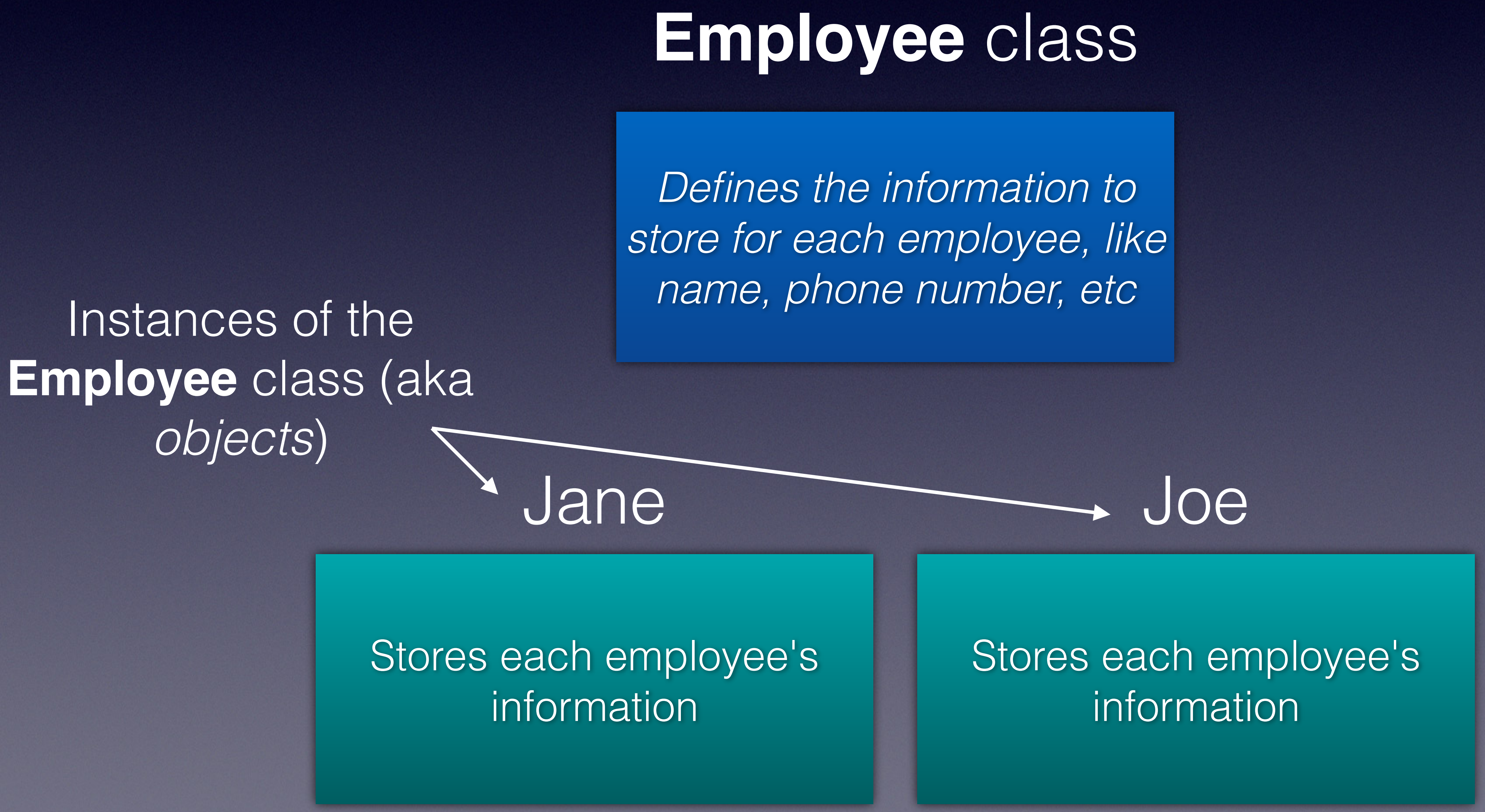
```
def filter(lf, lst):
    result = []
    for n in lst:
        if (x > 5):
            result.append(n)
    return result

print(filter(lambda x: x % 2 == 0, lst))
print(filter(lambda x: x > 5, lst))
```

Object Oriented Programming in Python

- OOP focuses on creating reusable code
- A **class** is a blueprint for the **object**
    - When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated
- An object (instance) is an instantiation of a class.

An Example

- You want to store information of each employee in your company



Employee class

Defines the information to store for each employee, like name, phone number, etc

Instances of the **Employee** class (aka *objects*)

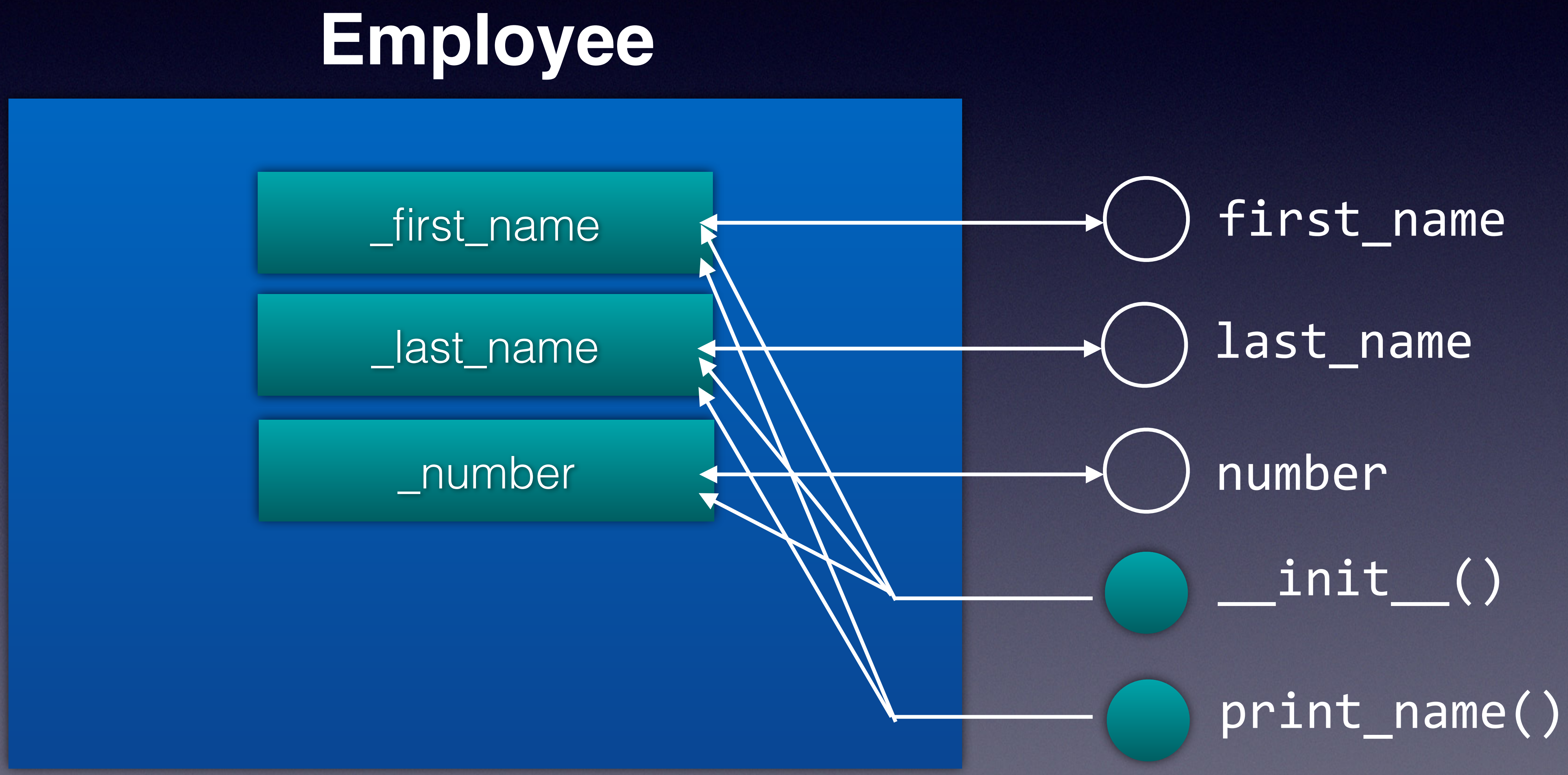
Jane

Stores each employee's information

Joe

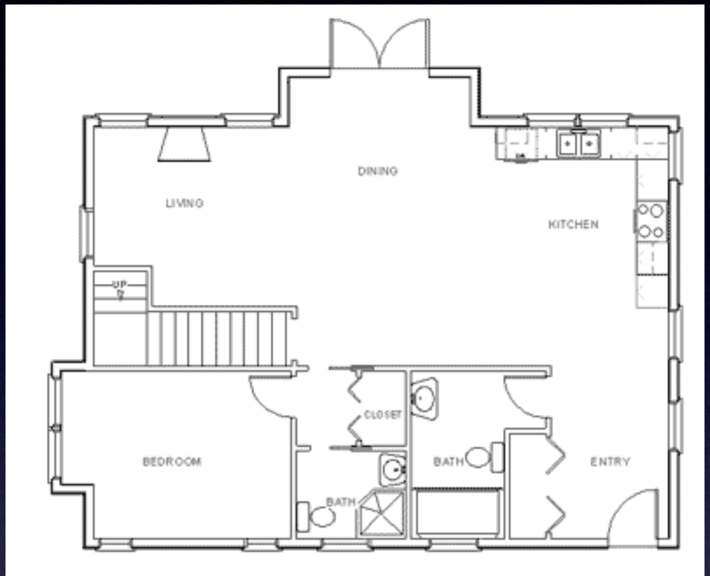
Stores each employee's information

Class


Employee
_first_name
_last_name
_number
first_name
last_name
number
__init__()
print_name()

Class and Objects

Class



Object

Object

Object

Properties

Iterators and Generators

Iteration

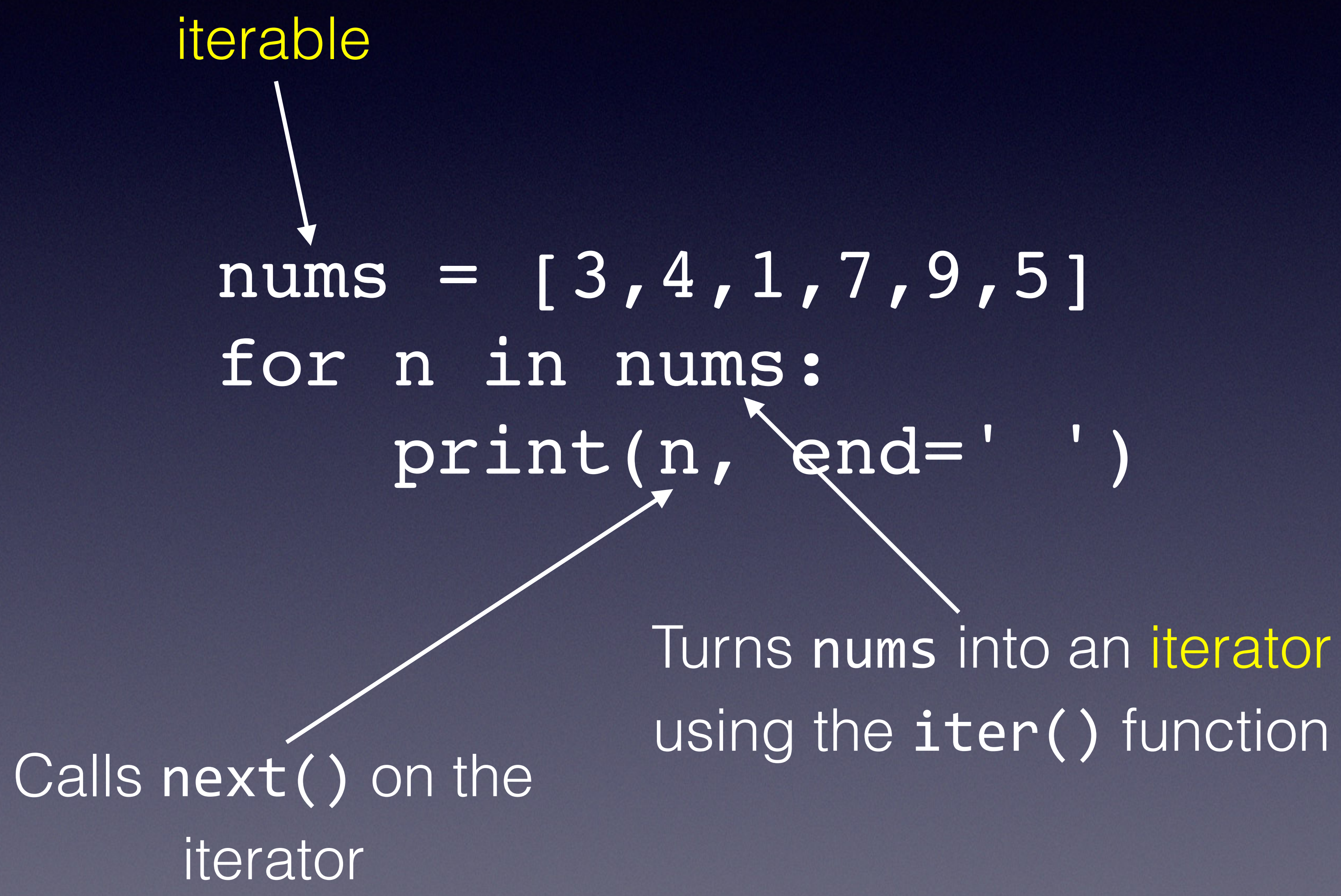
- Iteration means obtaining an item from something, one item at a time
- In Python, a number of built-in types support iterations, e.g. list, str, etc
- A String is an *iterable*

```
s = "Python"
for c in s:
    print(c)
```

Range Object

```
r = range(5)
print(r)
for i in r:
    print(i)


i = iter(r)
print(next(i))
print(next(i))
print(next(i))
print(next(i))
print(next(i))
```

Iterator vs Iterable



iterable

```
nums = [3,4,1,7,9,5]
for n in nums:
    print(n, end=' ')
```

Turns **`nums`** into an iterator using the **`iter()`** function

Calls **`next()`** on the iterator

Generator

- You implement an iterator object using a **generator**

```
def goodies():
    yield "Cupcake"
    yield "Donut"
    yield "Eclair"
    yield "Froyo"
    yield "Gingerbread"
    yield "Honeycomb"
    yield "Ice Cream Sandwich"
    yield "Jelly Bean"
    yield "KitKat"
    yield "Lollipop"
    yield "Marshmallow"
```

```
# method 1
dessert = iter(goodies())
print(next(dessert))
print(next(dessert))
print(next(dessert))
```

```
# method 2
for dessert in goodies():
    print(dessert)
```

Implementing Fibonacci using Generator

```
def fib(max):
    a, b = 0, 1
    while a < max:
        yield a
        a, b = b, a + b

# method 1
for i in fib(10):
    print (i, end=" ")

# method 2
i = iter(fib(10))
print(next(i)) # 0
print(next(i)) # 1
print(next(i)) # 1
print(next(i)) # 2
```

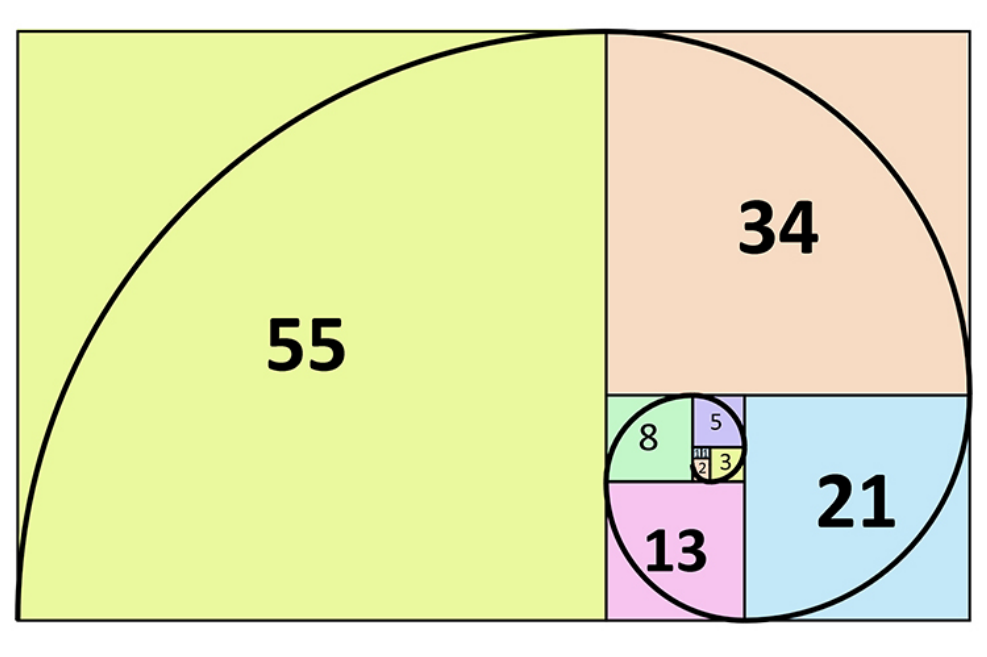
Uses of Generators

- Generators are good for calculating large sets of results where you don't know if you are going to need all results, or where you don't want to allocate the memory for all results at the same time
  - Generators are memory efficient since they only require memory for the one value they yield
  - Generators are useful for generating values ad infinitum

Fibonacci

- Consider the following example of generating a fibonacci sequence:

```
def fib(n):
    result = []
    a, b = 0, 1
    while b < n:
        result.append(a)
        a, b = b, a + b
    return result

print(fib(10))
```



Infinite Fibonacci

- What if you need to generate an infinite sequence of Fibonacci sequence?
- You could do something like this:

```
def fib():
    result = []
    a, b = 0, 1
    while True:
        result.append(a)
        a, b = b, a + b
    return result

print(fib())
```

← Infinite loop!

Infinite Fibonacci

- You could use a generator to do that!

```
def fib():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

i = iter(fib())
print(next(i))
print(next(i))

# OR

for c in fib():
    print(c)
    input()
```

Modules

Modules

```
def do_something1():
    …

def do_something2():
    …

def do_something3():
    …
```

A module is a .py file containing functions

Python Module Search Path

When you import a module named `hello`, the interpreter will first search for a built-in module called `hello`. If a built-in module is not found, the Python interpreter will then search for a file named `hello.py` in a list of directories that it receives from the `sys.path` variable.

The following commands print out the path searched by Python when you import a module

```
$ python

>>> import sys

>>> sys.path
```

Checking Modules Installed

See all modules installed

```
$ python

>>> help('modules')
```

Get help on specific module, e.g. requests

```
$ python

>>> help('requests')
```

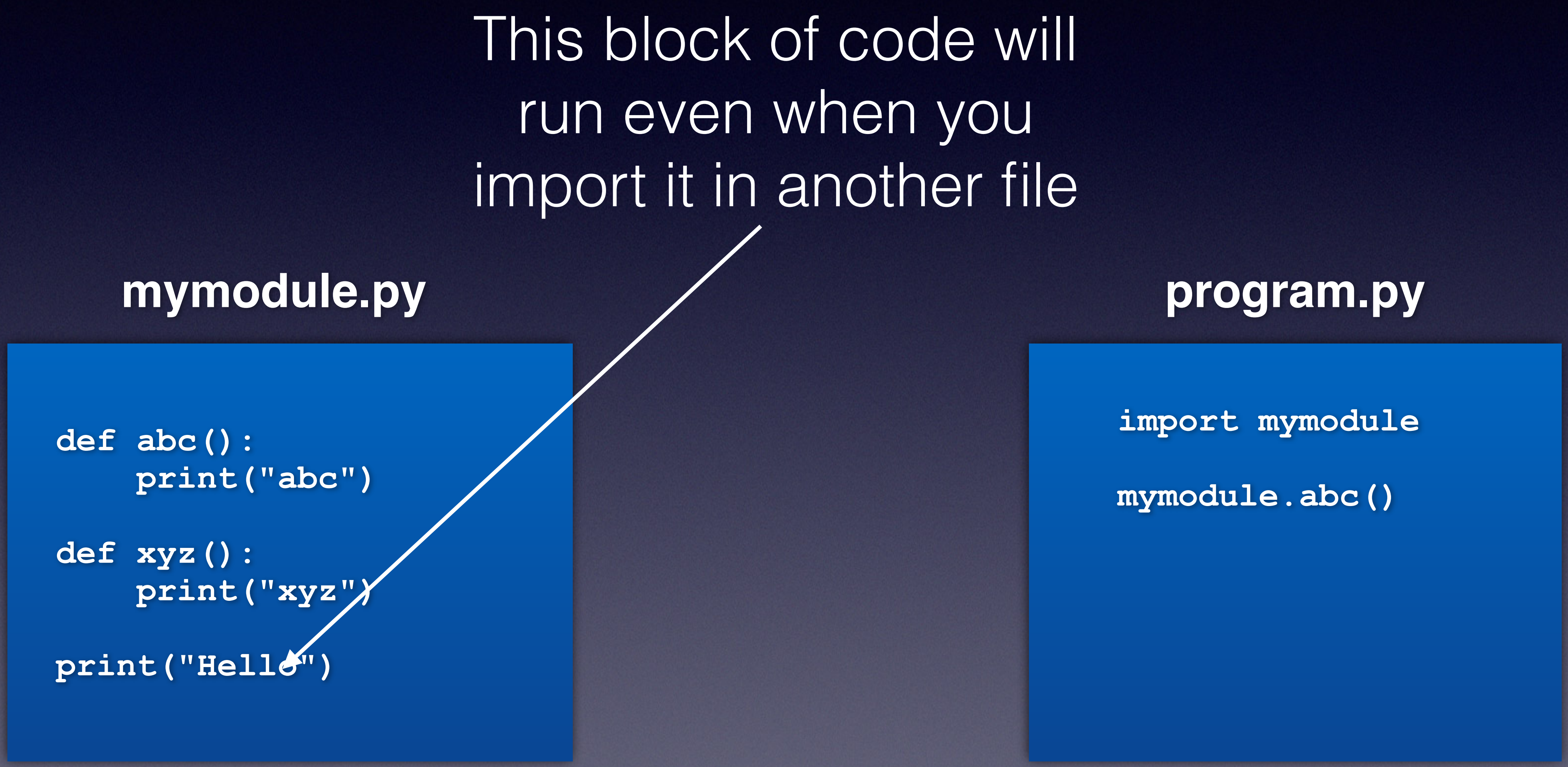
Importing Modules in Python

- `import X`
  - imports module X; use *X.name* to refer to functions in module X
- `import X as Y`
  - imports module X and rename it as Y; use *Y.name* to refer to functions in module X
- `from X import *`
  - imports the module X; use *name* directly to refer to functions in module X
- `from X import a, b, c`
  - imports the module X; use *a*, *b*, or *c* directly to refer to the functions in module X
- `from X import a as d`
  - imports the module X; renames the *a* function as *d* and use *d* directly to refer to the function in module X

mymodule.py

- Download file from
  - `https://pastebin.com/raw/jYAcTyMa`

Importing Modules



This block of code will run even when you import it in another file

mymodule.py

```
def abc():
    print("abc")

def xyz():
    print("xyz")

print("Hello")
```

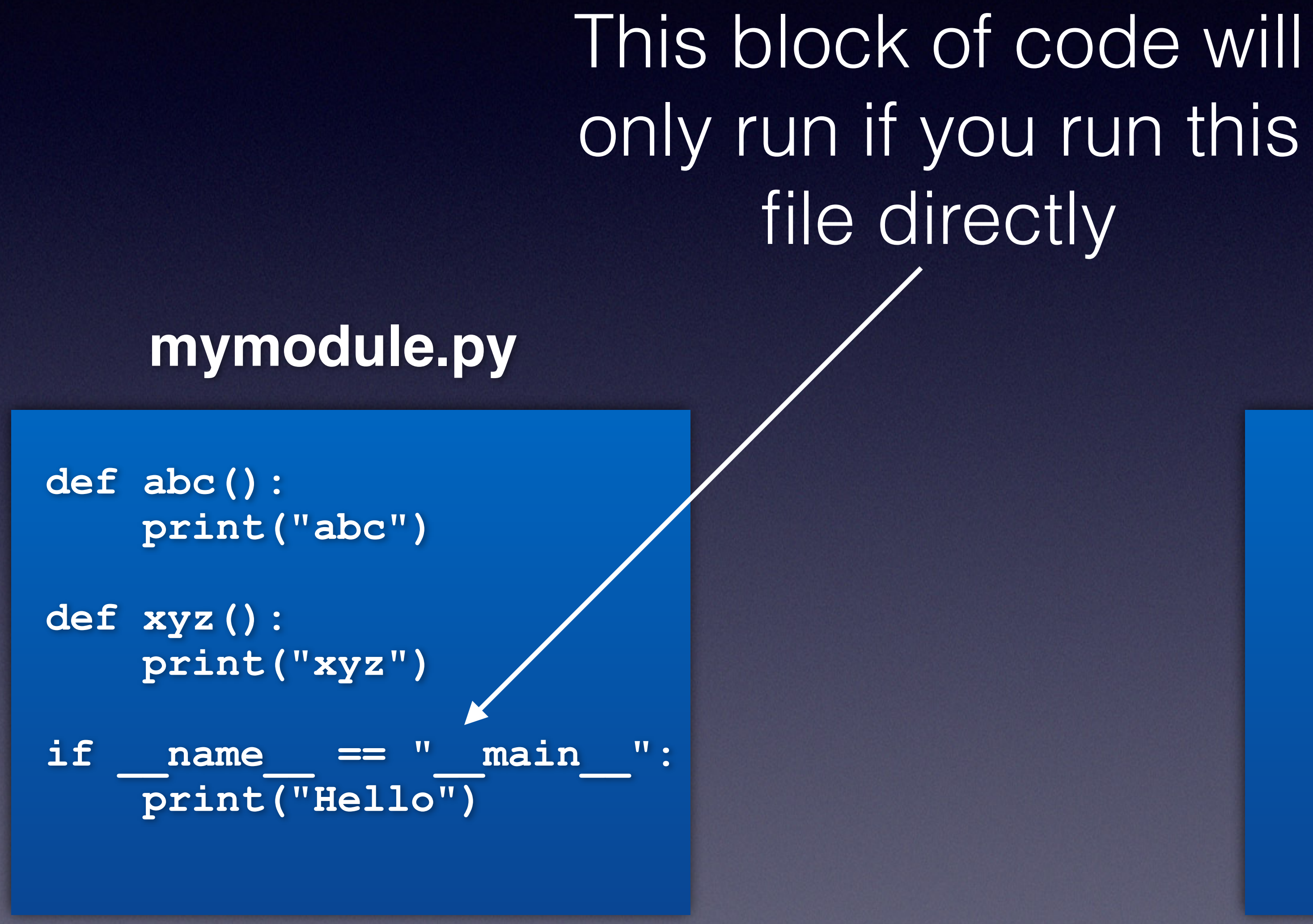
program.py

```
import mymodule

mymodule.abc()
```

```
$ python mymodule.py
Hello
```

```
$ python program.py
Hello
abc
```

if __name__ == "__main__"



This block of code will only run if you run this file directly

mymodule.py

```
def abc():
    print("abc")

def xyz():
    print("xyz")

if __name__ == "__main__":
    print("Hello")
```

```
$ python mymodule.py
Hello
```

program.py

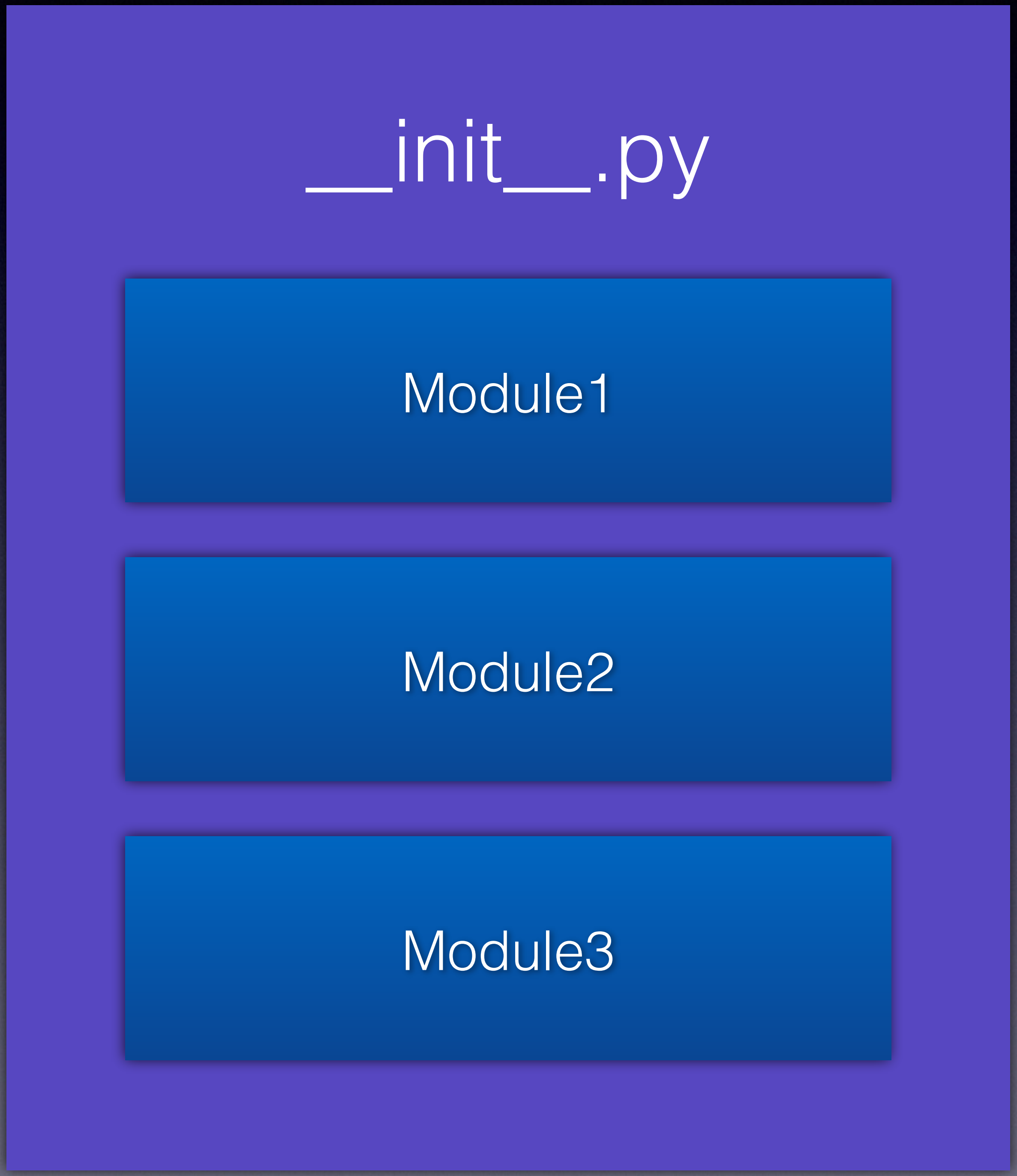
```
import mymodule

mymodule.abc()
```

```
$ python program.py
abc
```

Python Packages

Packages

- A Python package is a directory of Python modules
  - A __init__.py file is required to make Python treat the directory as containing a package



Installing Packages using pip

- **pip** is the standard package manager for Python
- Installing Packages
  - **`pip install <package_name>`**`, e.g.` *`pip install flask`*
- Version
  - `pip --version`
- Find the location of installed package
  - **`pip show`** ***`<package_name>`***`, e.g.` *`pip show flask`*
- By default, all packages are installed globally
  - Use the **`--user`** option to install for the current user, e.g. *`pip install --user flask`*

Quick Recap

- Lambda Functions
- Classes and objects
- Iterators and Generators
- Modules
- Packages

NumPy

What is NumPy?

- NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.
  - Written using a mixture of C and Python
    - Heavy lifting part written using C

Problems with Python List

- The key problem with the Python’s list data type is its efficiency
  - *list* allows you to have non-uniform type items - each item in the list are stored in a memory location, with the list containing an “array” of pointers to each of these locations
- A Python list requires:
  - At least 4 bytes per pointer
  - At least 16 bytes for the smallest Python object – 4 bytes for pointer, 4 bytes for reference count, 4 bytes for the value, and these round up to 16 bytes.
- And because of the way Python list is implemented, accessing items in a large list is computationally expensive.

NumPy Array

- In NumPy, an array is of type `ndarray` (n-dimensional array)
  - all elements are of the **same** type

```
import numpy as np

l1 = [1,2,3,4,5]
array1 = np.array(l1)  # rank 1 array
print (array1)         # [1 2 3 4 5]
print (array1.shape)   # (5,)

print (array1[0])      # 1
print (array1[1])      # 2
print (array1[1:3])    # [2 3]
print (array1[:-2])    # [1 2 3]
print (array1[3:])     # [4 5]
```

Boolean Array Indexing

- Consider the following array:

```
nums = np.array([23,45,78,89,23,11,22])
```

- How do you retrieve all the even numbers in the array?

- Using *boolean array indexing*, you could do the following:

```
even_nums = nums % 2 == 0
print (even_nums)
# [False False  True False False False  True]

print (nums[even_nums])
# [78 22]
```

```
print (nums[nums % 2 == 0])
```

Another Example

```
prices = np.array([45,23,56,89,12,48])
```

- Print out all the prices that are between 20 and 50

```
reasonable = (prices >= 20) & (prices <= 50)
print prices[reasonable]
# [45 23 48]
```

Array Math

```
x1 = np.array([[1,2,3],[4,5,6]])
y1 = np.array([[7,8,9],[2,3,4]])

print x1 + y1     # same as np.add(x1,y1)
'''
[[ 8 10 12]
 [ 6  8 10]]
'''

print x1 - y1     # same as np.subtract(x1,y1)
print x1 * y1     # same as np.multiply(x1,y1)
print x1 / y1     # same as np.divide(x1,y1)
```

Cumulative Sums

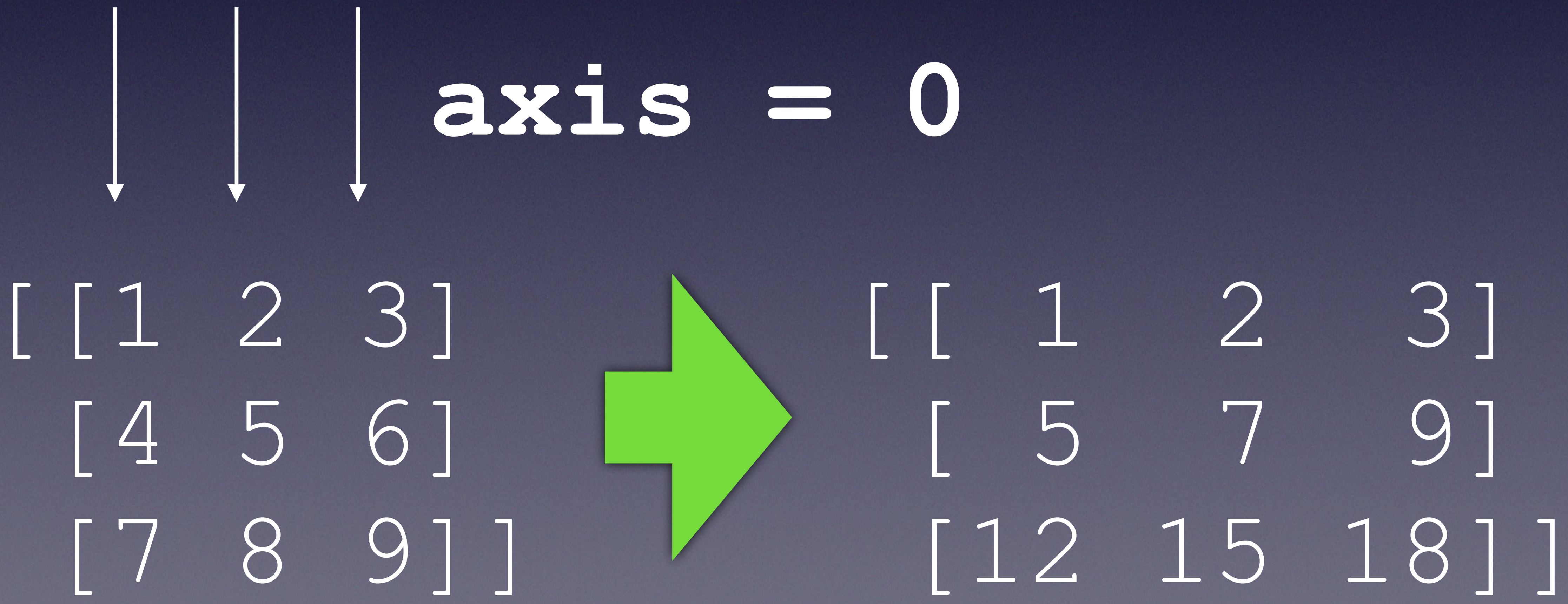
```
a = np.array([(1,2,3), (4,5,6), (7,8,9)])
print (a)
'''
[[1 2 3]
 [4 5 6]
 [7 8 9]]
'''
```

- To generate the cumulative sum of all the numbers in the array, use the **`cumsum()`** function

```
print (a.cumsum())   # prints the cumulative sum of all
                     # the elements in the array
'''
[ 1  3  6 10 15 21 28 36 45]
'''
```

Cumulative Sums

```
print (a.cumsum(axis=0))   # sum over rows for each of
                           # the 3 columns
'''
[[ 1  2  3]
 [ 5  7  9]
 [12 15 18]]
'''
```



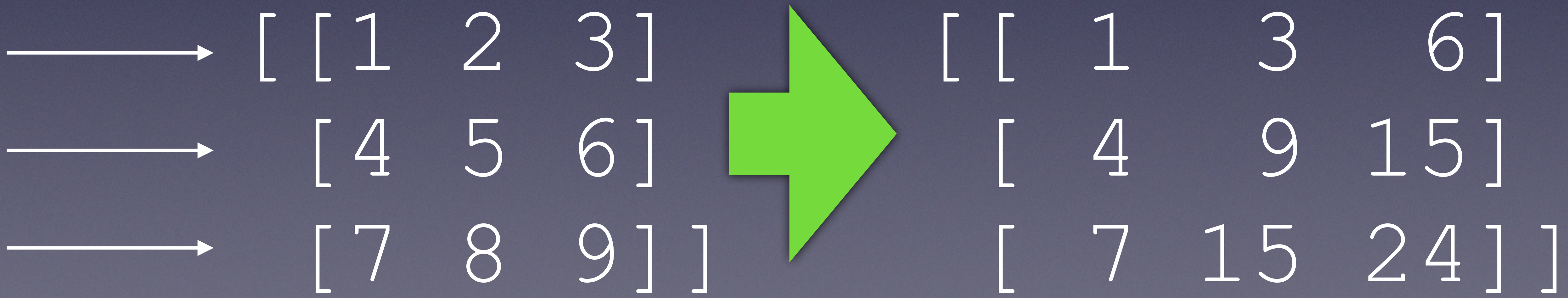
axis = 0

```
[[1 2 3]        [[ 1  2  3]
 [4 5 6]         [ 5  7  9]
 [7 8 9]]        [12 15 18]]
```

Cumulative Sums

```
print (a.cumsum(axis=1))   # sum over columns for each
                           # of the 3 rows
'''
[[ 1  3  6]
 [ 4  9 15]
 [ 7 15 24]]
'''
```

axis = 1



```
[[1 2 3]        [[ 1  3  6]
 [4 5 6]         [ 4  9 15]
 [7 8 9]]        [ 7 15 24]]
```

Pandas

Pandas

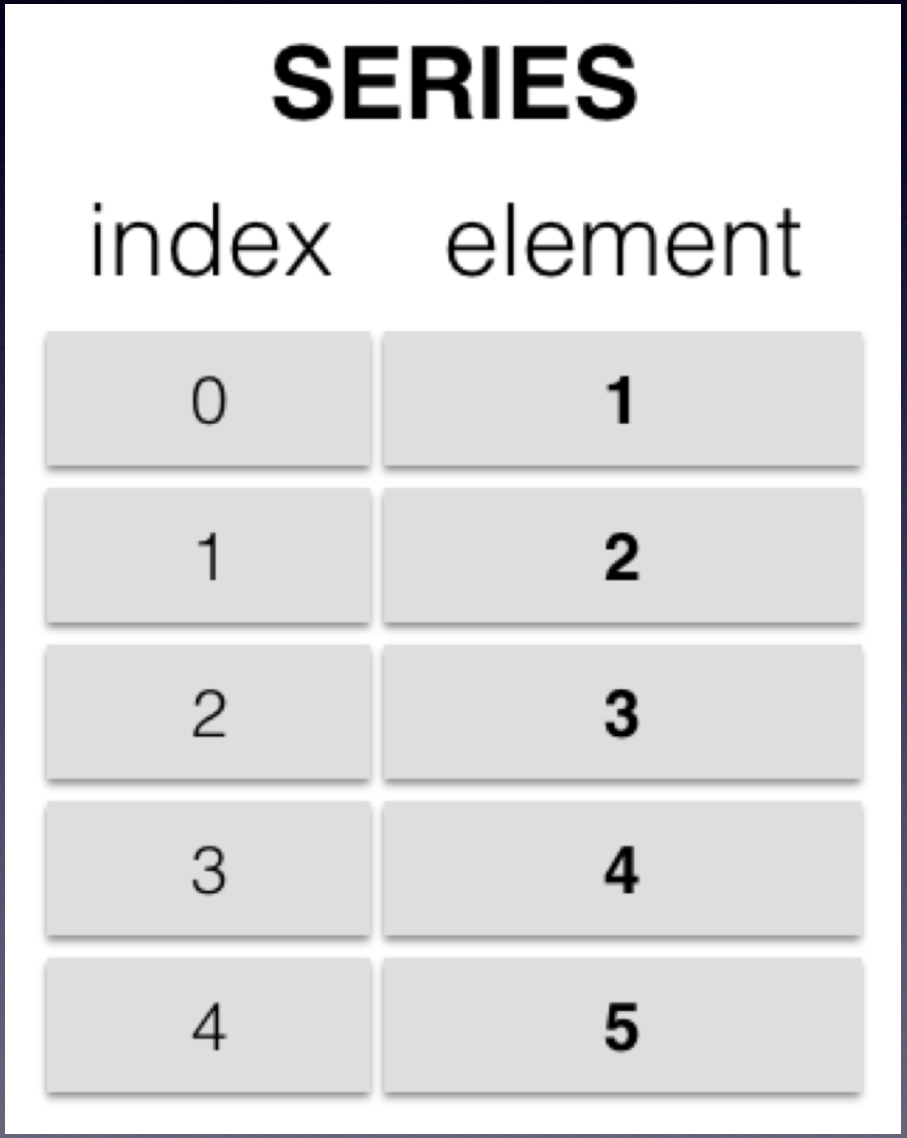
- While Python supports lists and dictionaries for manipulating structured data, it is not well suited for manipulating numerical **tables**, such as the those stored in CSV files
- As such, you should use **Pandas**
    - stands for **Pan**el **D**ata **A**naly**s**is
    - Pandas is a software library written for Python for data manipulation and analysis

Key Data Structures in Pandas

- **Series**
  - A Series is a one-dimensional NumPy-like array, with each element having an index (0, 1, 2, … by default); a Series behaves like a dictionary, with an index
- **DataFrame**
  - A DataFrame is a two-dimensional NumPy-like array; think of it as a table

Series

```
import pandas as pd

series = pd.Series([1,2,3,4,5])
print series
'''
0    1
1    2
2    3
3    4
4    5
dtype: int64
'''
```



SERIES

index	element
0	**1**
1	**2**
2	**3**
3	**4**
4	**5**

Specifying Your Own Index in a Series

```
import pandas as pd

series = pd.Series([1,2,3,4,5],
                   ['a','b','c','d','e'])
print series
'''
a    1
b    2
c    3
d    4
e    5
dtype: int64
'''
```

Generating Date Ranges

```
import pandas as pd

dates1 = pd.date_range('20160525', periods=12)
print dates1
'''
DatetimeIndex(['2016-05-25', '2016-05-26',
               '2016-05-27', '2016-05-28',
               '2016-05-29', '2016-05-30',
               '2016-05-31', '2016-06-01',
               '2016-06-02', '2016-06-03',
               '2016-06-04', '2016-06-05'],
              dtype='datetime64[ns]', freq='D')
'''
```

Default frequency is Day

Monthly Frequency

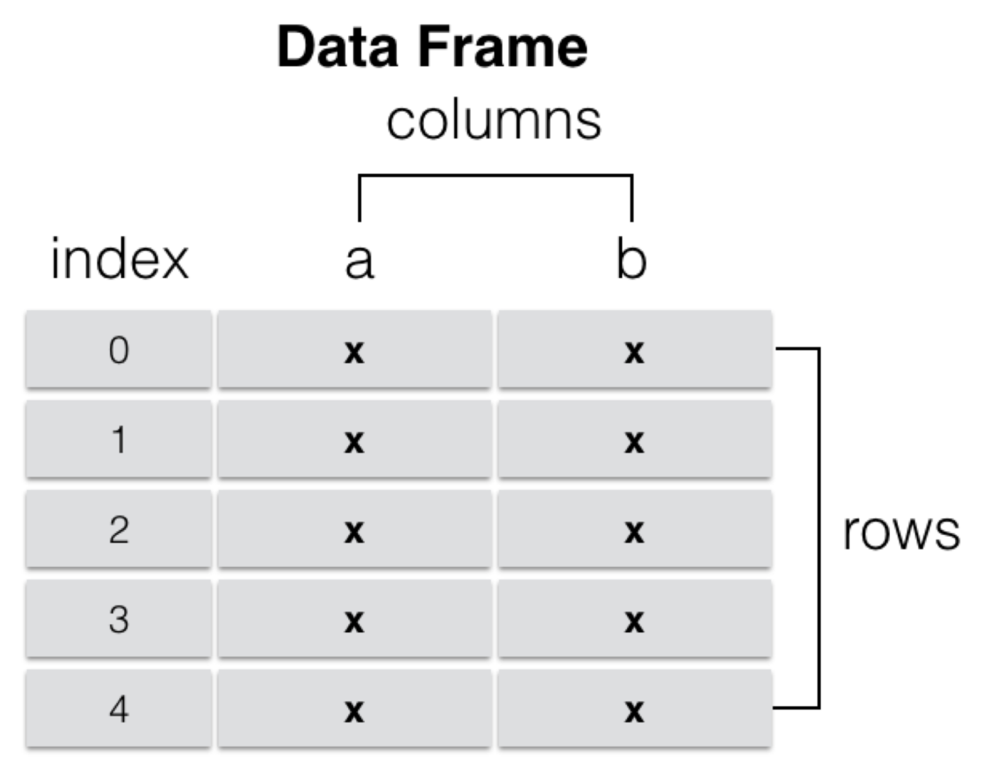
```
dates2 = pd.date_range('2016-05-01', periods=12,
                       freq='M')
print dates2
'''
DatetimeIndex(['2016-05-31', '2016-06-30',
               '2016-07-31', '2016-08-31',
               '2016-09-30', '2016-10-31',
               '2016-11-30', '2016-12-31',
               '2017-01-31', '2017-02-28',
               '2017-03-31', '2017-04-30'],
              dtype='datetime64[ns]', freq='M')
'''
```

Hourly Frequency

```
dates3 = pd.date_range('2016/05/17 09:00:00',
                        periods=8,
                        freq='H')
print dates3
'''
DatetimeIndex(['2016-05-17 09:00:00',
               '2016-05-17 10:00:00',
               '2016-05-17 11:00:00',
               '2016-05-17 12:00:00',
               '2016-05-17 13:00:00',
               '2016-05-17 14:00:00',
               '2016-05-17 15:00:00',
               '2016-05-17 16:00:00'],
              dtype='datetime64[ns]', freq='H')
'''
```

DataFrame

```
import pandas as pd
import numpy as np

data_frame = pd.DataFrame(np.random.randn(10,4),
                          columns=list('ABCD'))
print data_frame
'''
          A         B         C         D
0 -0.280362  1.428287 -0.258593  0.576082
1  2.049914  0.218698  1.622331  1.439911
2 -0.228631  0.554902 -1.002514  0.867128
3  0.651200  0.064399  0.943215 -0.305893
4 -1.667361  0.491074  0.687449  0.173527
5 -2.253975 -1.761170  0.706407  0.247967
6  0.627704  0.651399 -1.589874  1.511602
7 -0.193086  0.330129 -0.574044 -0.652339
8 -0.101104  0.087196 -0.972024  1.695285
9 -1.279870 -0.078746 -1.600358  1.676298
'''
```



Data Frame

columns

rows

index	a	b
0	x	x
1	x	x
2	x	x
3	x	x
4	x	x

Using a Date Range as the Index of a DataFrame

```
days = pd.date_range('20150525', periods=10)
data_frame = pd.DataFrame(np.random.randn(10,4),
                          index=days,
                          columns=list('ABCD'))
print data_frame

'''
                   A         B         C         D
2015-05-25 -0.181824 -0.522341 -0.629486 -0.098926
2015-05-26 -0.786451  0.270572 -0.007755  0.407279
2015-05-27 -1.801745 -0.627653  0.017884 -0.294941
2015-05-28 -0.199777 -0.343533 -0.847143  0.230196
2015-05-29 -0.470902 -1.882163  1.589637  0.041875
2015-05-30  0.223365 -0.367830  0.901914 -1.574907
2015-05-31 -0.701686  2.185077 -0.787870 -1.014857
2015-06-01  2.078889  0.467649  0.462715  0.731940
2015-06-02 -0.739564  0.055060 -0.414679  1.229497
2015-06-03  1.086807  0.134102 -1.114484 -0.277467
'''
```

Selecting Data From DataFrames

```
print data_frame['A']             # prints column A

'''
2015-05-25    0.400942
2015-05-26    0.553610
2015-05-27   -1.772219
2015-05-28    0.298267
2015-05-29   -0.079830
2015-05-30    0.619363
2015-05-31   -0.217129
2015-06-01   -0.111042
2015-06-02    1.080578
2015-06-03    1.937649
'''
```

Slicing DataFrame

Slicing by index

```
# prints rows with index from 2015-05-25 to 2015-05-28
print data_frame['2015-05-25':'2015-05-28']
'''
                   A         B         C         D
2015-05-25  0.400942  0.734476 -0.900102 -0.148904
2015-05-26  0.553610  1.729898  1.248708  0.353235
2015-05-27 -1.772219 -2.182172 -0.439986 -1.672310
2015-05-28  0.298267  1.049802 -2.093472  1.330577
'''
```

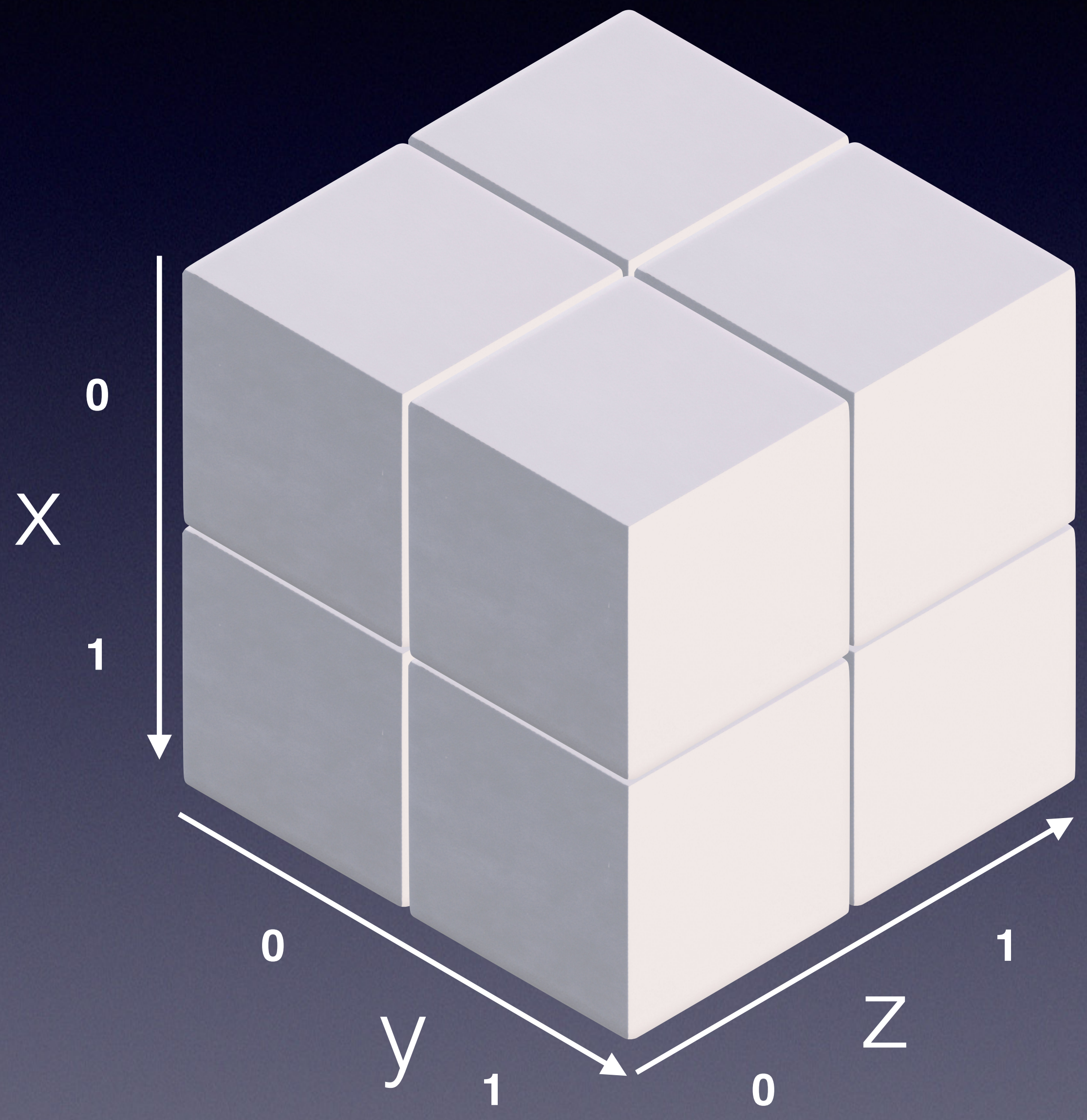
Slicing by row number

```
print data_frame[2:5]      # prints row 3 through row 5
'''
                   A         B         C         D
2015-05-27 -1.772219 -2.182172 -0.439986 -1.672310
2015-05-28  0.298267  1.049802 -2.093472  1.330577
2015-05-29 -0.079830  1.169019  0.047177 -0.599912
'''
```

Transposing DataFrame

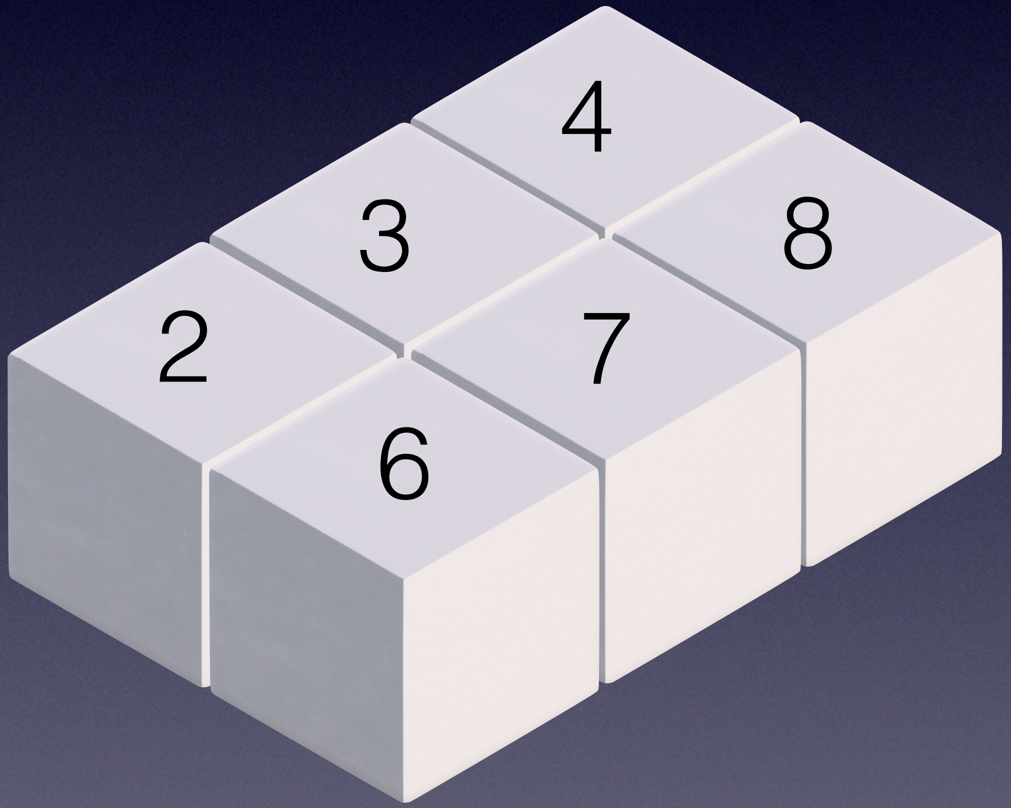
```
print data_frame.T
'''
   2015-05-25  2015-05-26  2015-05-27  2015-05-28  2015-05-29  2015-05-30
A    0.575812   -0.401051   -1.767028    1.148867  -1.013309   -0.232075
B   -0.994374   -0.225347   -0.683786    1.600078  0.655725    0.210781
C   -0.641241    1.003547    0.308813    1.066649  -0.181266    0.140533
D   -0.384547    0.256077   -0.980992    0.647792  0.151229    0.260636

   2015-05-31  2015-06-01  2015-06-02  2015-06-03
A   -0.892355   -0.178719   -0.174579   -0.364807
B   -1.073592    0.985476   -1.347515   -0.735336
C   -0.260684   -0.706353   -0.872690    1.385756
D    1.456331   -1.800571    0.416017   -0.392111
'''
```

Visualizing a 3D Array



- Shape - `(x,y,z)`

Shape (1,2,3)

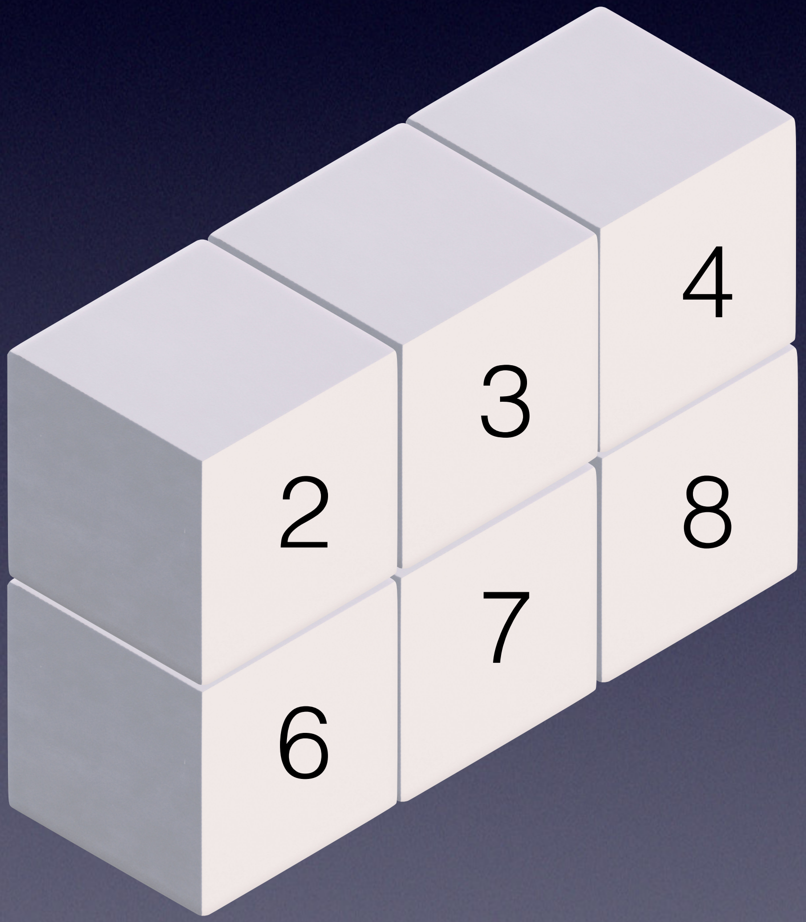


- Shape - (1,2,3)

- Value -

```
[[[2 3 4]
  [6 7 8]]]
```

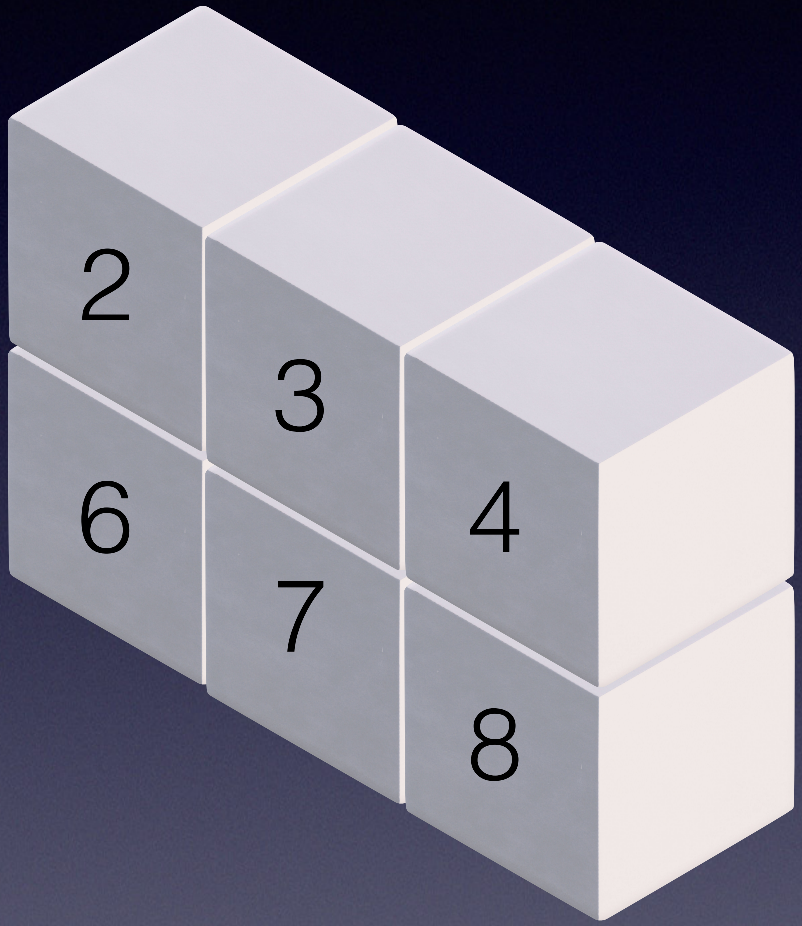
Shape (2,1,3)



- Shape - (2,1,3)
- Value -

```
[[[2 3 4]]

 [[6 7 8]]]
```

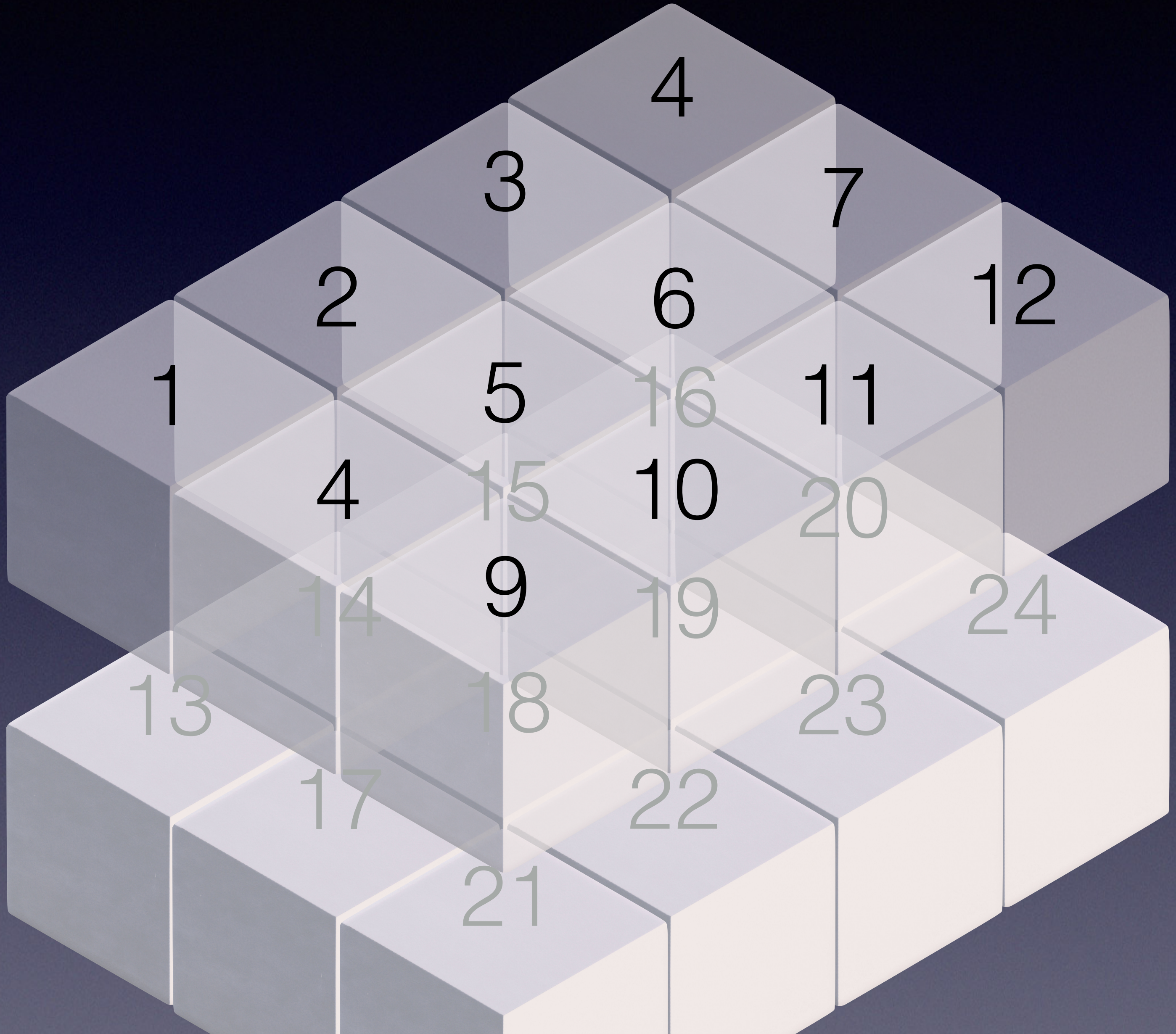
Shape (2,3,1)



- Shape - (2,3,1)
- Value -

```
[[[2]
  [3]
  [4]]

 [[6]
  [7]
  [8]]]
```

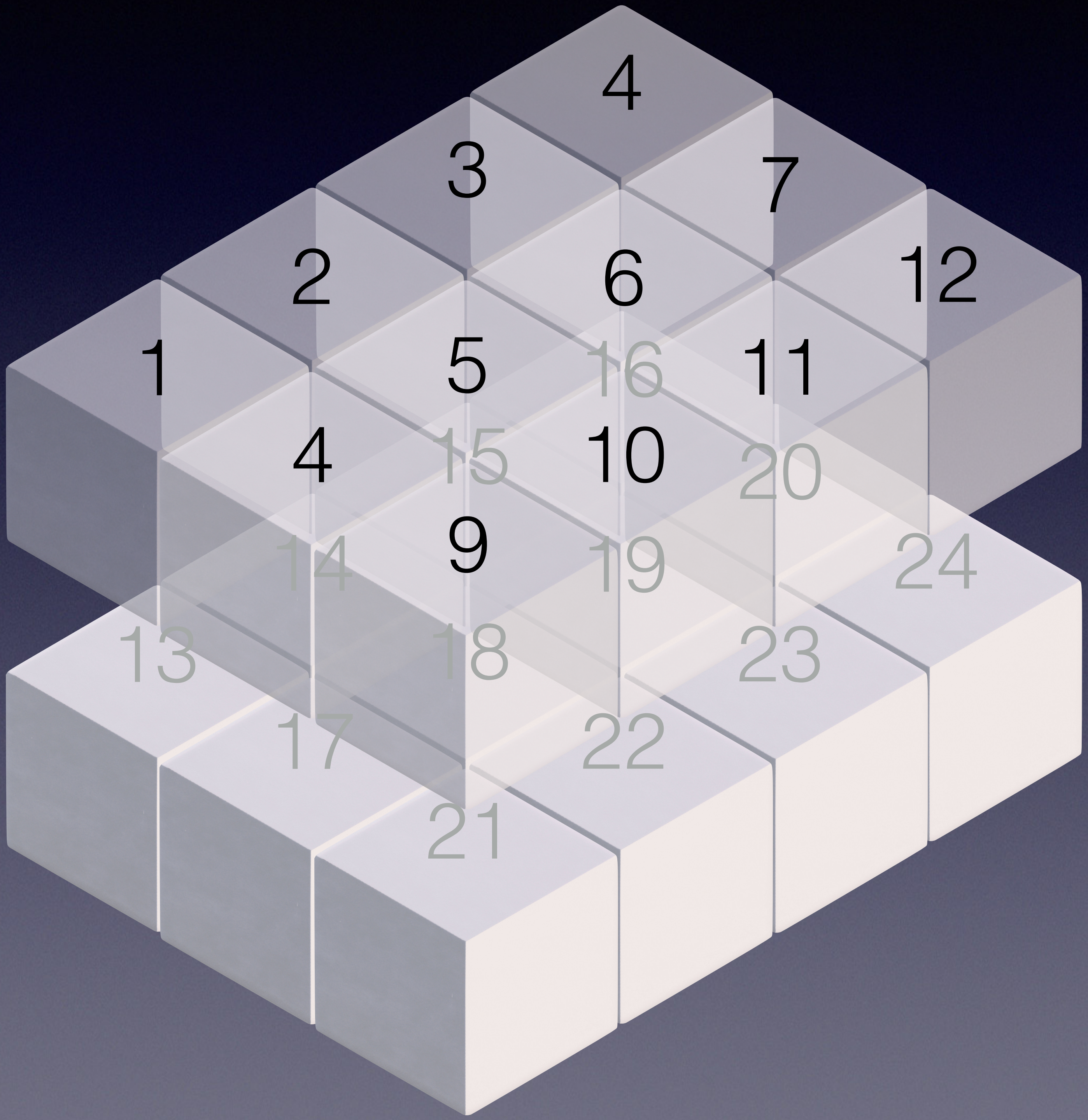
Shape (2,3,4)



- Shape - (2,3,4)
- Value -

```
[[[ 1  2  3  4]
  [ 4  5  6  7]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 18 19 20]
  [21 22 23 24]]]
```

Indexing Example



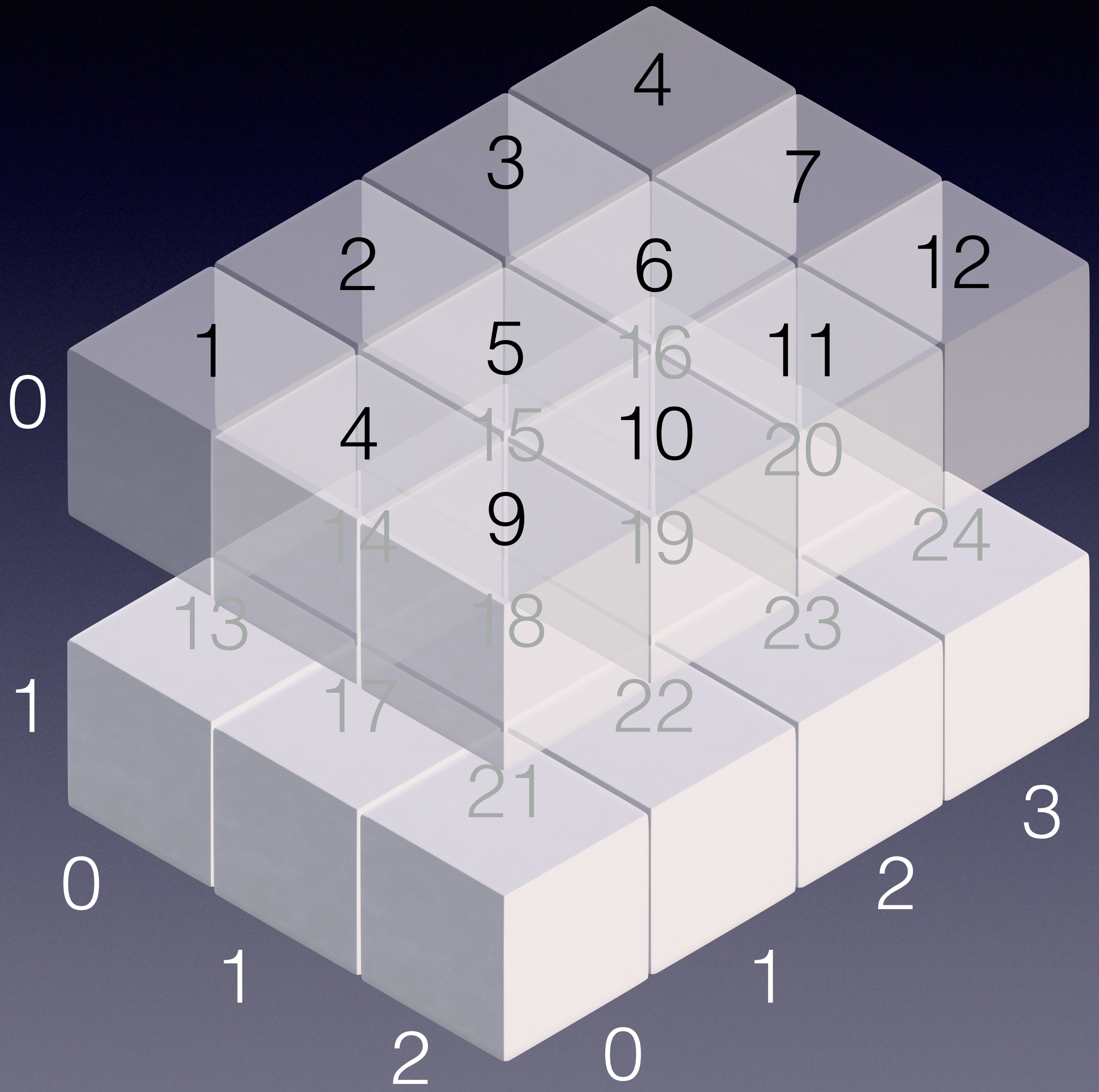
```
arr = np.array([[[1,2,3,4],
                 [4,5,6,7],
                 [9,10,11,12]],
                [[13,14,15,16],
                 [17,18,19,20],
                 [21,22,23,24]]])
```

```
arr[1]
```

```
arr[1,1]
```

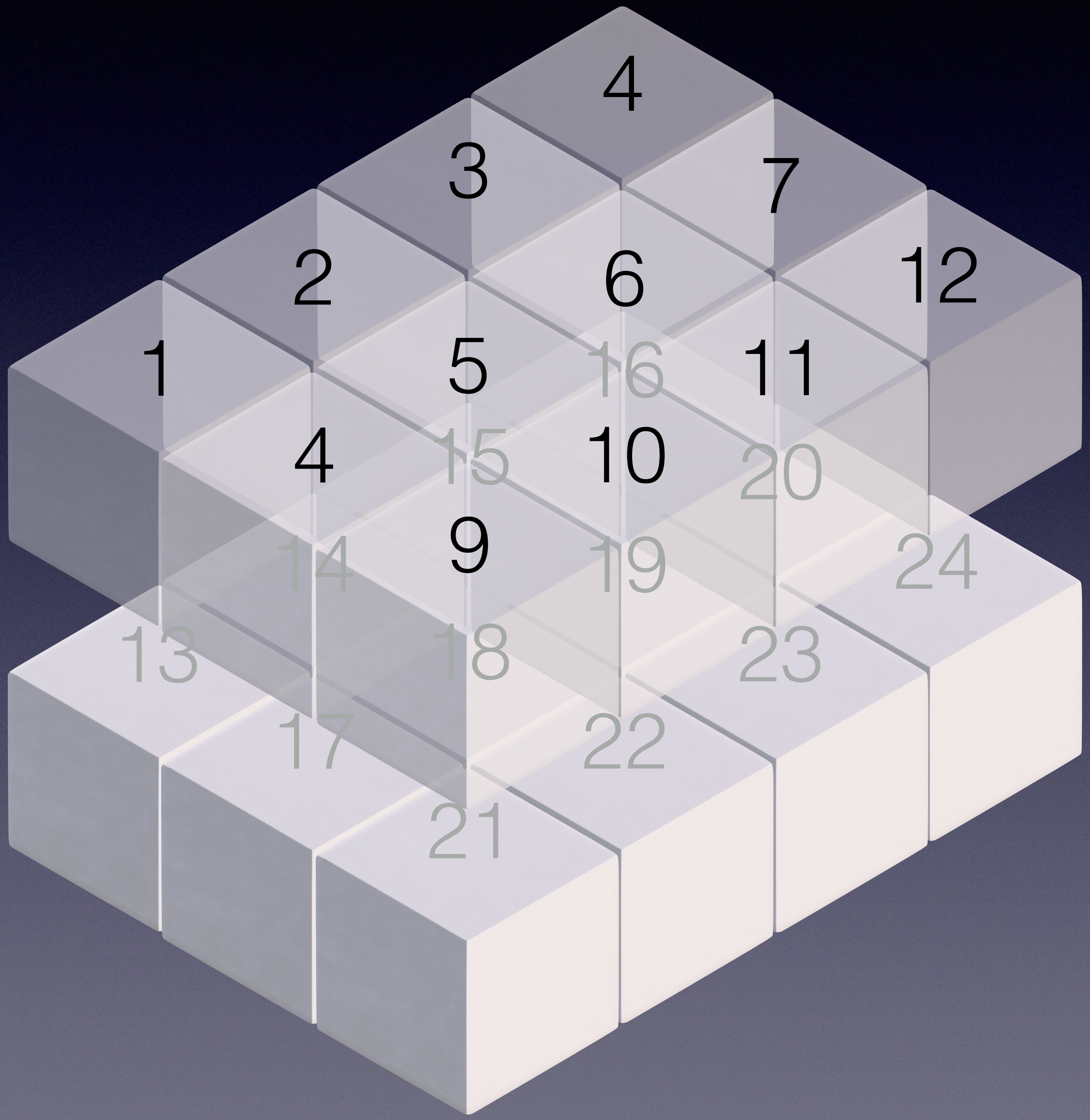
```
arr[1,1,1]
```

Indexing (Answer)



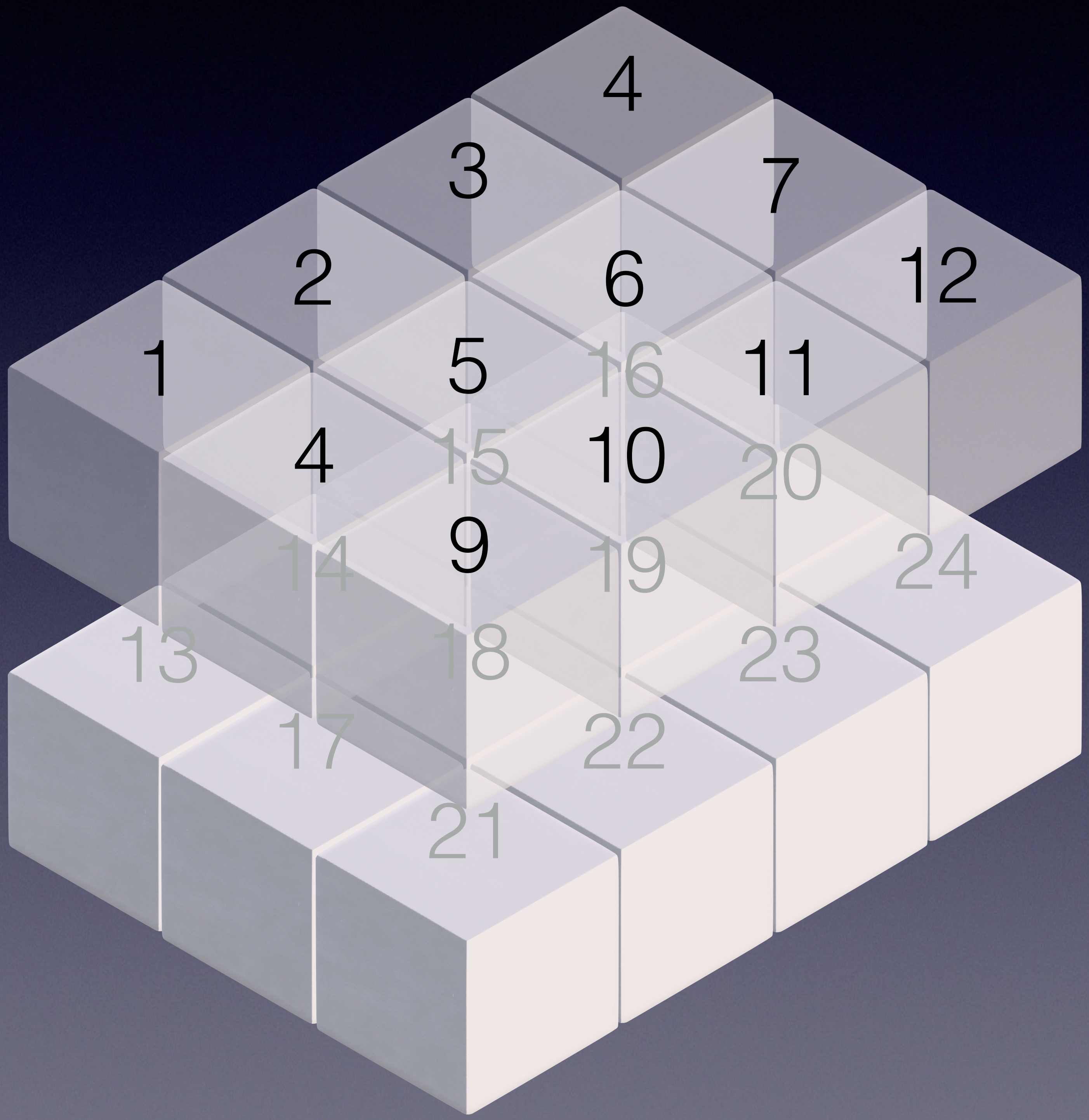
```
arr = np.array([[[1,2,3,4],
                 [4,5,6,7],
                 [9,10,11,12]],
                [[13,14,15,16],
                 [17,18,19,20],
                 [21,22,23,24]]])


arr[1]
'''
array([[13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]])
'''


arr[1,1] # array([17, 18, 19, 20])

arr[1,1,1] # 18
```

Slicing Example



```
arr[0:2]

arr[0:2,1]

arr[0:2,1:3,3]
```

Slicing (Answer)

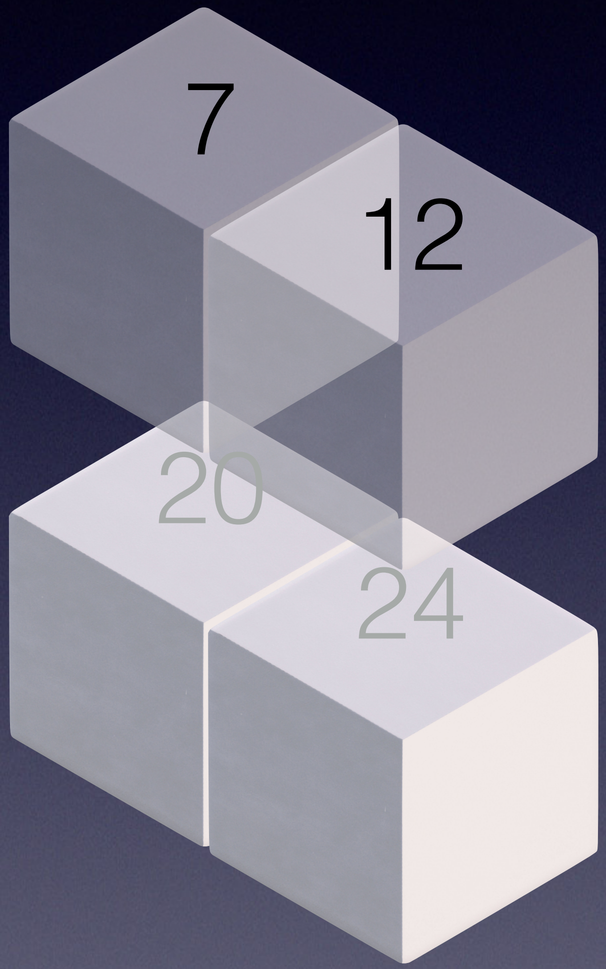


```
arr[0:2]
'''
array([[[ 1,  2,  3,  4],
        [ 4,  5,  6,  7],
        [ 9, 10, 11, 12]],

       [[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]]])
'''
```

```
arr[0:2,1]
'''
array([[ 4,  5,  6,  7],
       [17, 18, 19, 20]])
'''
```

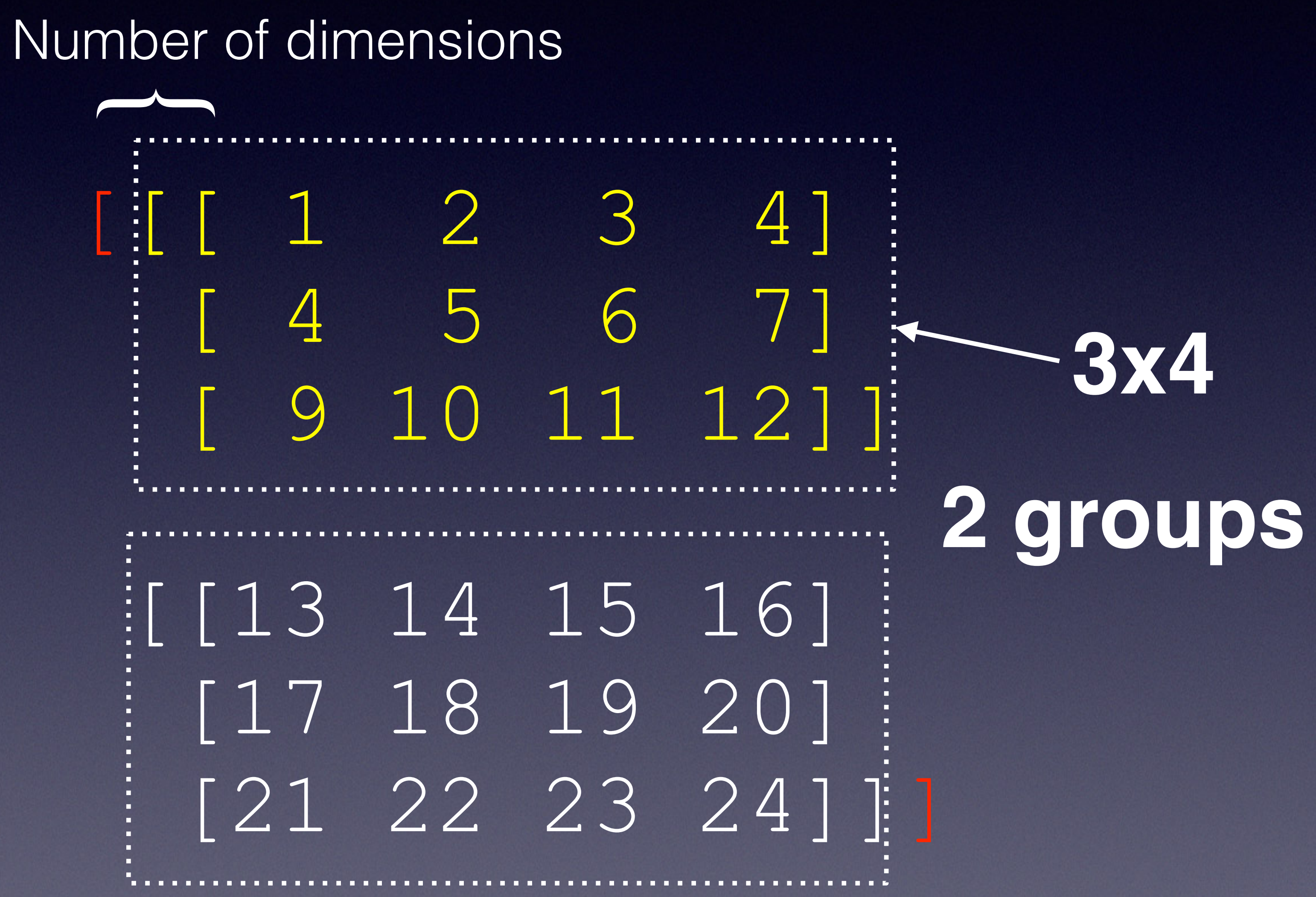
```
arr[0:2,1:3,3]
'''
array([[ 7, 12],
       [20, 24]])
'''
```

Slicing Example



```
arr[0:2,1:3,3]
'''
array([[ 7, 12],
       [20, 24]])
'''
```

Determining the Shape of Array Based on Outputs



Number of dimensions

```
[[[ 1  2  3  4]
  [ 4  5  6  7]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 18 19 20]
  [21 22 23 24]]]
```

3x4

2 groups

- Count the number of dimensions
- Count the shape of inner most array group
  - Count the number of such groups
- Shape is (2,3,4)

Try It Out

```
[[[2]
  [3]
  [4]]

 [[6]
  [7]
  [8]]

 [[2]
  [3]
  [4]]

 [[6]
  [7]
  [8]]]
```

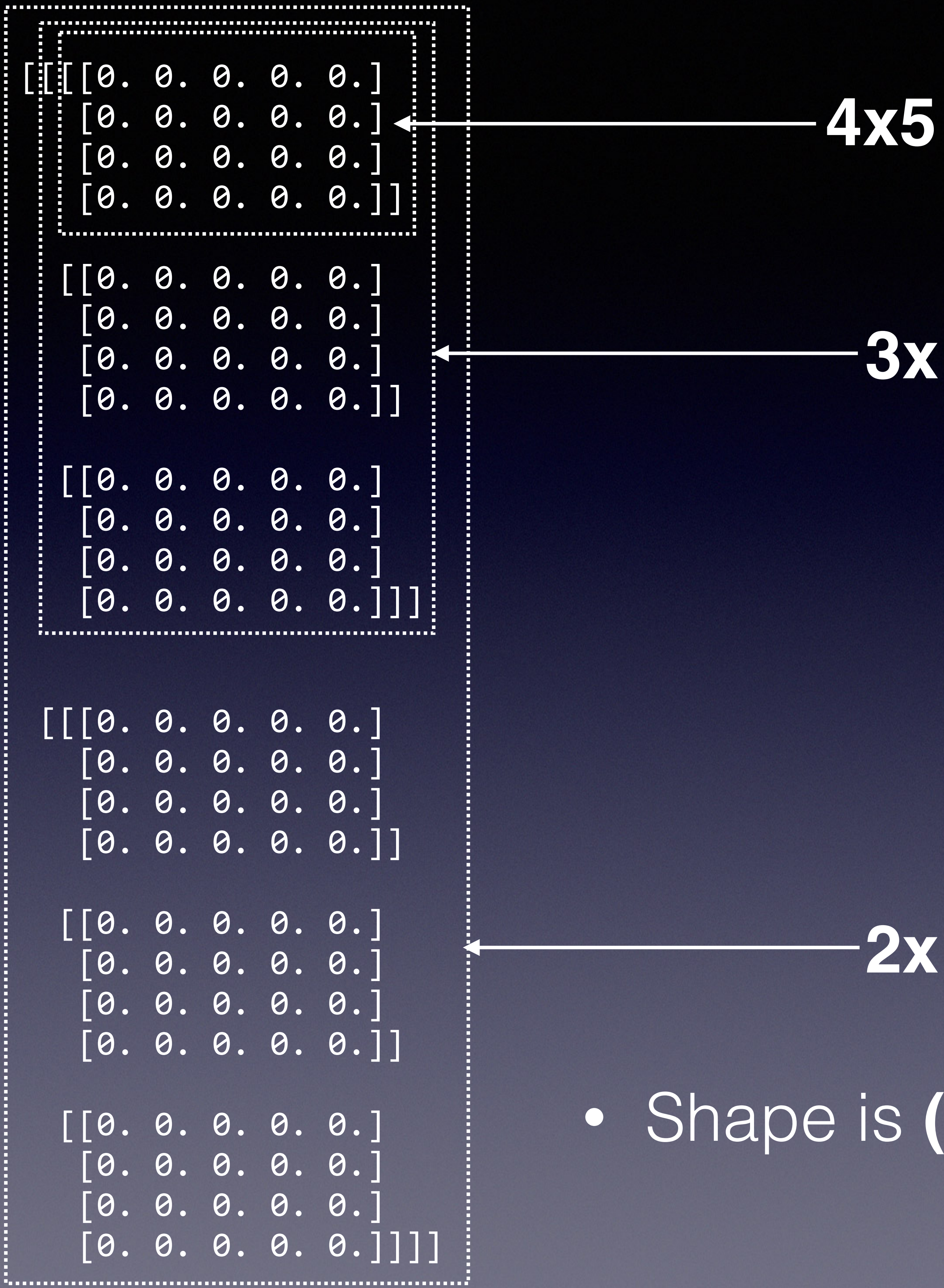
Answer

```
[[[2]
  [3]
  [4]]

 [[6]
  [7]
  [8]]

 [[2]
  [3]
  [4]]

 [[6]
  [7]
  [8]]]
```

- Count the shape of inner most array group
  - **3x1**
- Count the number of such groups
  - **4**
- Shape is (**4,3,1**)

Another Example

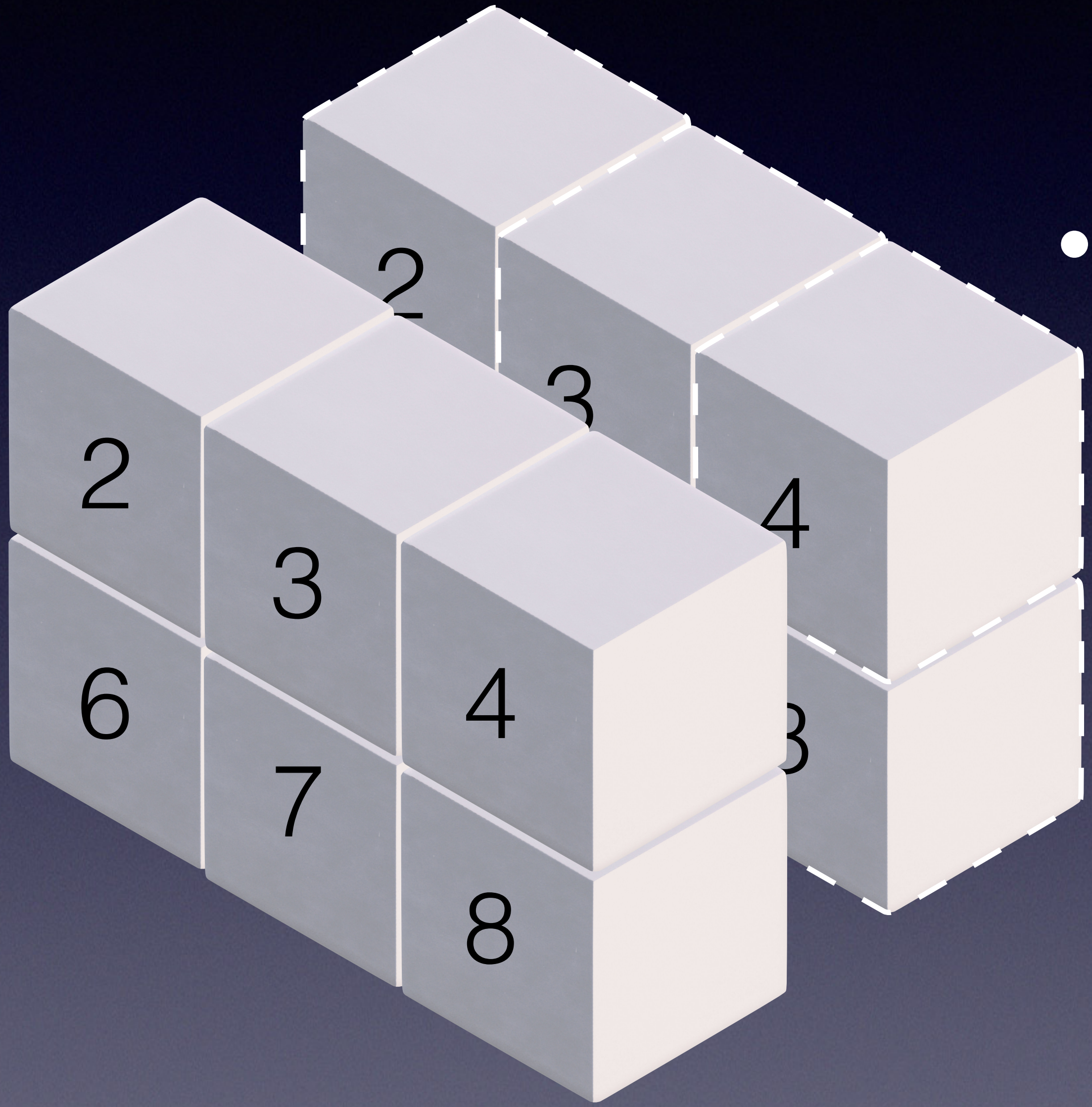
```
[[[[0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]]

  [[0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 0. 0. 0. 0.]
   [0. 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. 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. 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. 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.]]]


 [[[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. 0. 0. 0. 0.]]

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

- Shape is **(2,3,4,5)**

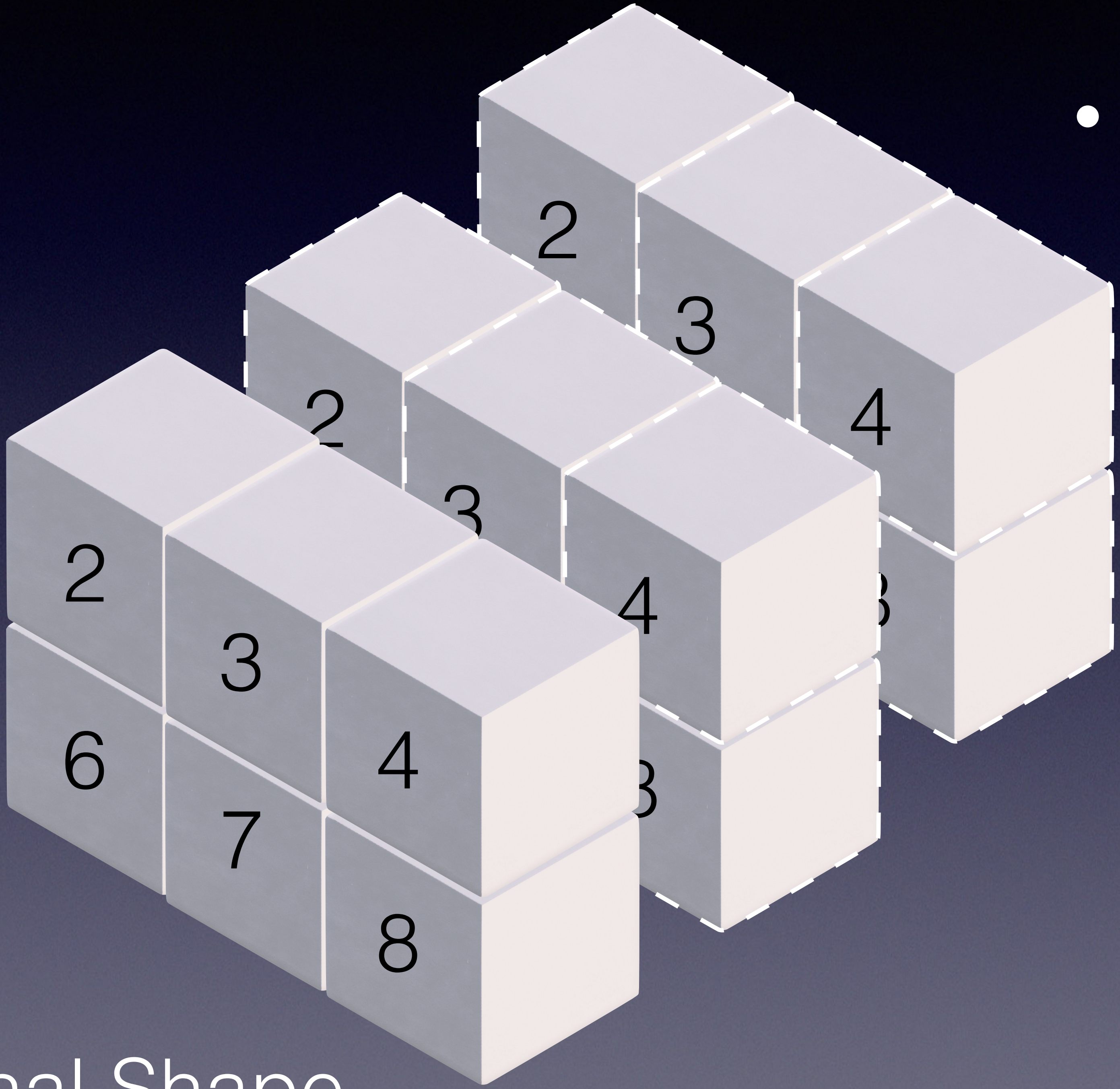
Broadcasting



- Broadcast to new shape (2,3,2)

- Original Shape - (2,3,1)

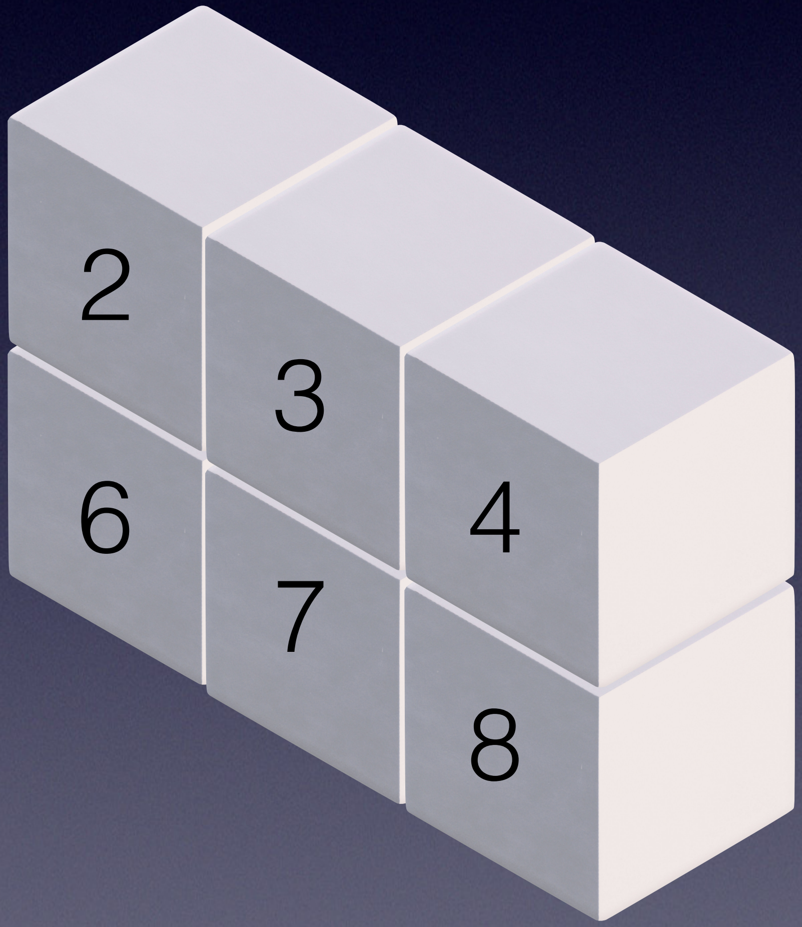
Broadcasting



- Broadcast to new shape (2,3,**3**)

- Original Shape - (2,3,1)

Broadcasting

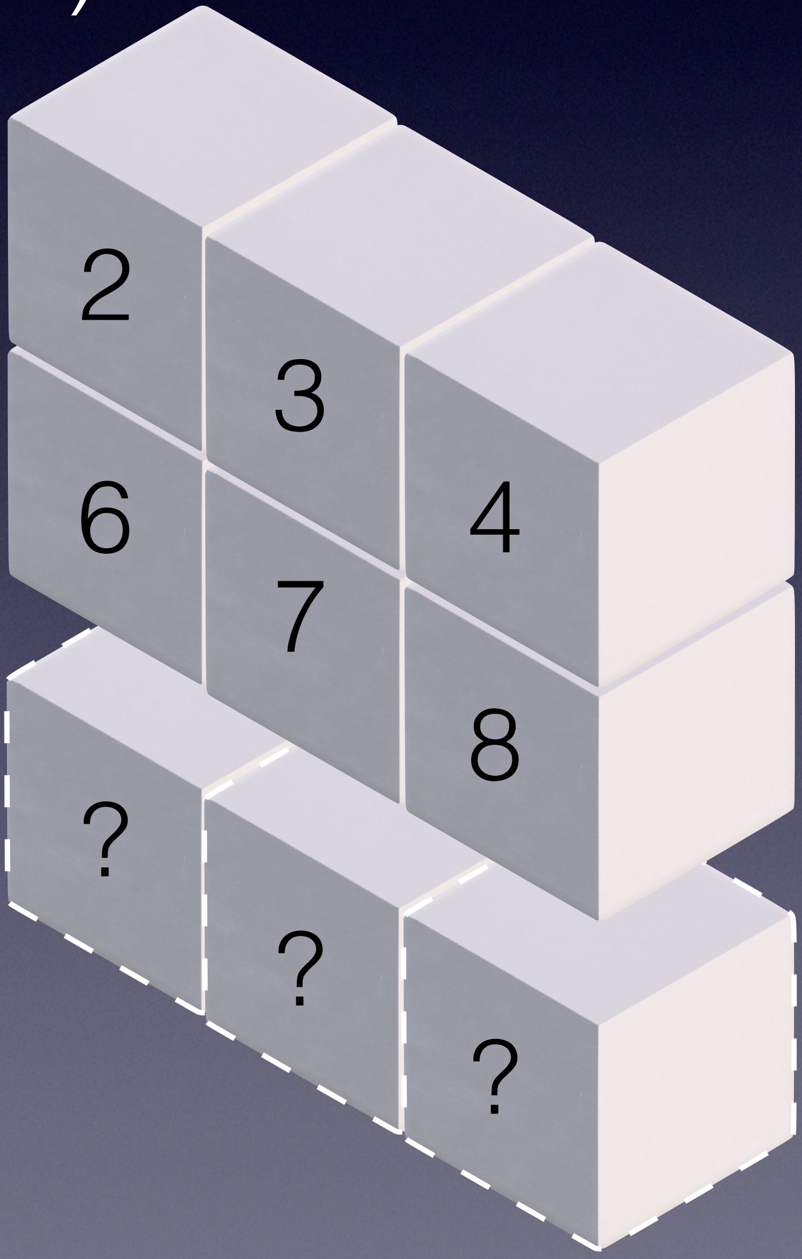


- Can you broadcast to new shape (3,3,1) ?

- Original Shape - (2,3,1)

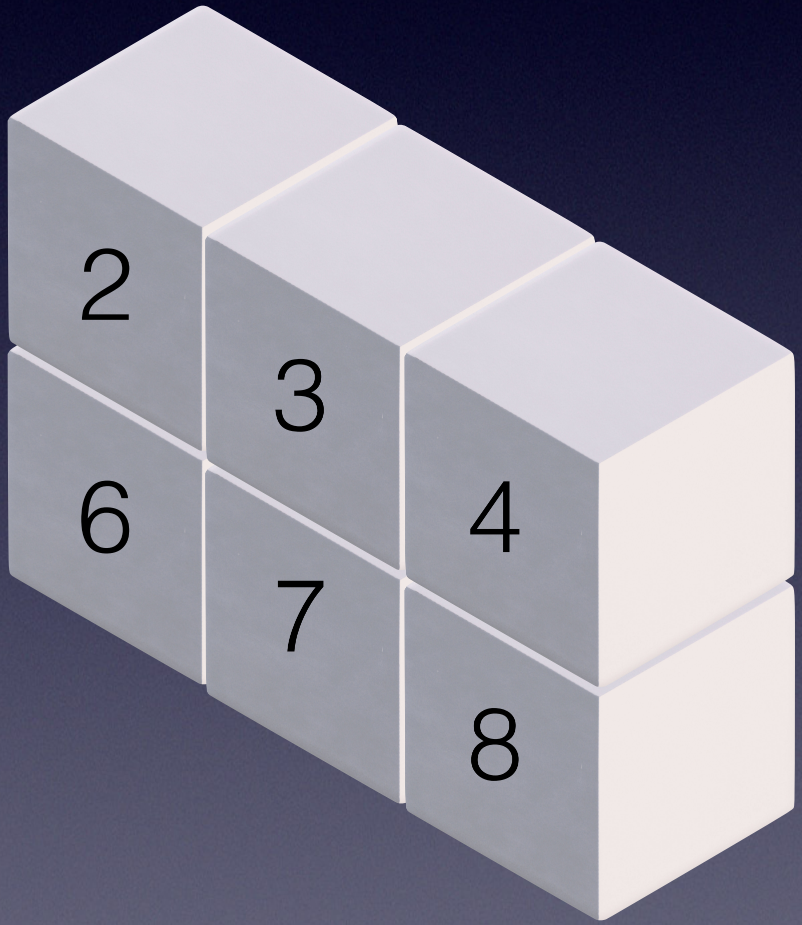
Broadcasting

- Original Shape - (2,3,1)



- Can you broadcast to new shape (3,3,1) ?
- **Error**

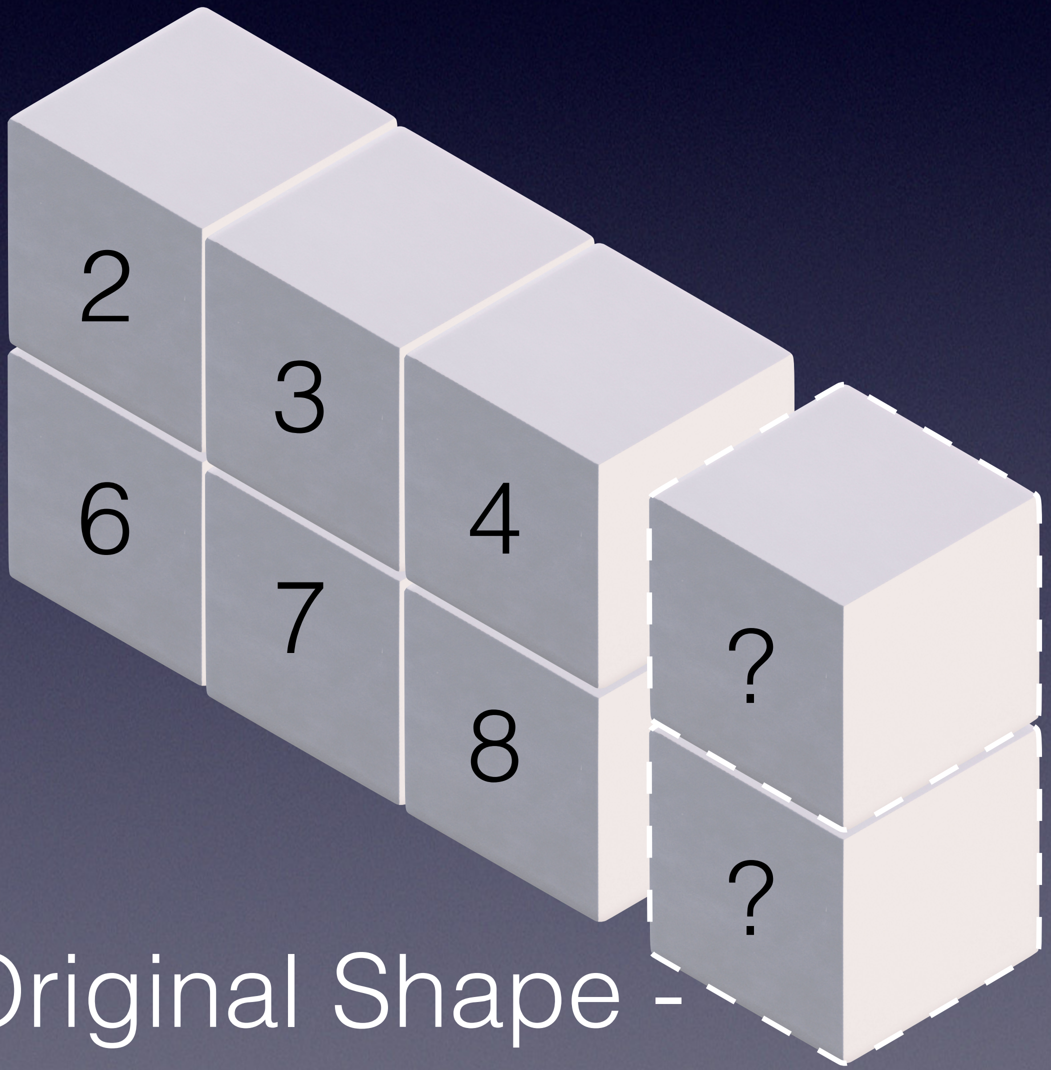
Broadcasting



- Original Shape - (2,3,1)

- Can you broadcast to new shape (2,4,1) ?

Broadcasting



- Can you broadcast to new shape (2,4,1) ?
- **Error**

- Original Shape - (2,3,1)

DataFrame CheatSheet

Column name `df["x"]`	Column names `df[["x","y"]]`	
Row numbers `df[0:2]`		
Row Numbers Column Numbers `df.iloc[1:2,1:2]`	Row Numbers Column Numbers `df.iloc[1:2,[1,2]]`	Row Numbers Column Numbers `df.iloc[[1,2],[1,2]]`
Index Values Column Values `df.loc[1:2,"x":"y"]`	Index Values Column Values `df.loc[1:2,["x","y"]]`	Index Values Column Values `df.loc[[1,2],["x","y"]]`
Index Values Column Numbers `df.loc[1:2, df.columns[0:2]]`		Index Values Column Numbers `df.loc[1:2, df.columns[[0,2]]]`

- Download at:
    - `https://bit.ly/39ob43b`

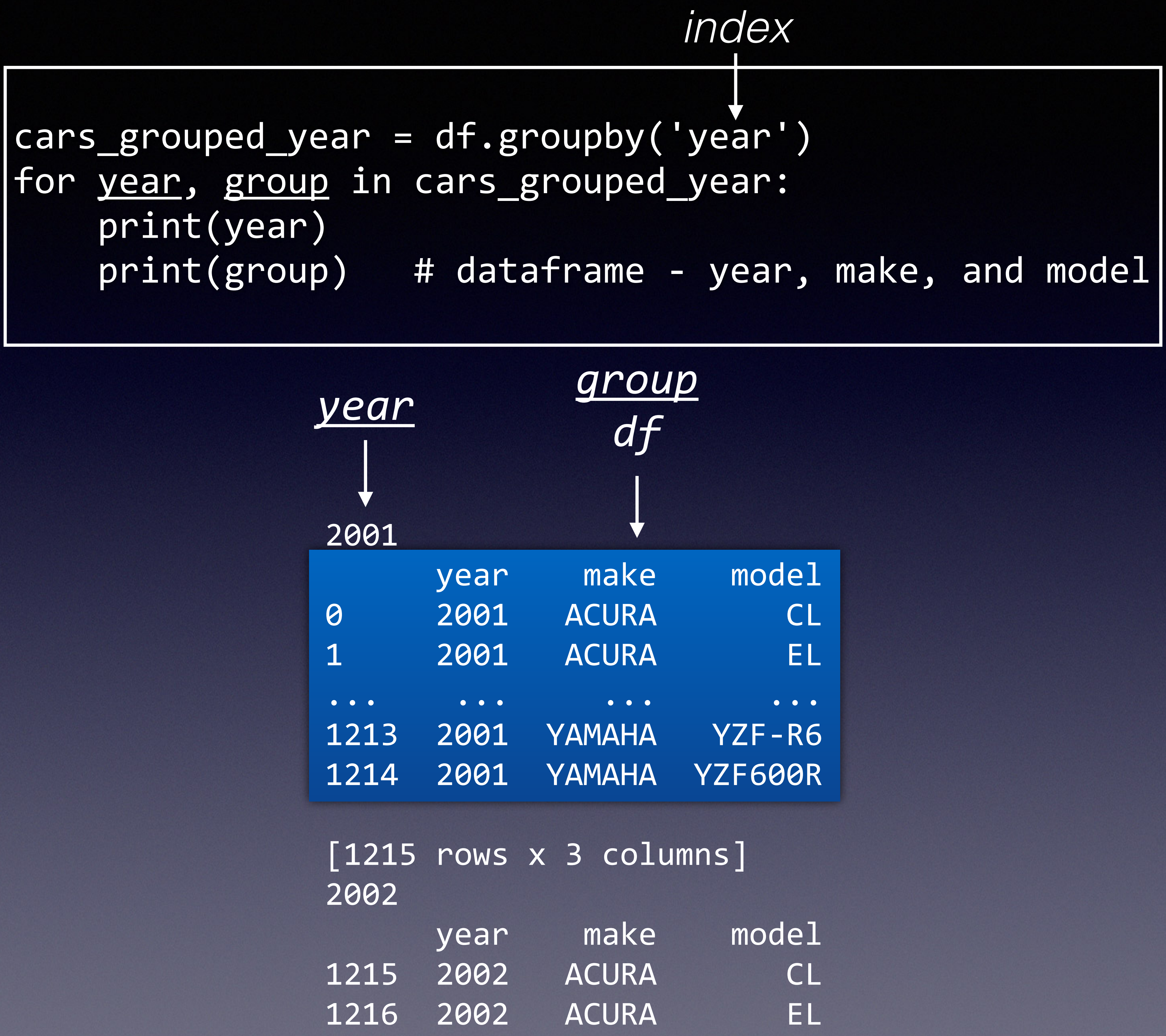
Developer Learning Solutions

http://calendar.learn2develop.net

GroupBy

Dataset

- Download dataset from:
  - `https://bit.ly/3dSCq3b`



index

```
cars_grouped_year = df.groupby('year')
for year, group in cars_grouped_year:
    print(year)
    print(group)   # dataframe - year, make, and model
```

year

group
df

```
2001
      year    make    model
0     2001   ACURA       CL
1     2001   ACURA       EL
...    ...     ...      ...
1213  2001  YAMAHA   YZF-R6
1214  2001  YAMAHA  YZF600R

[1215 rows x 3 columns]
2002
      year    make    model
1215  2002   ACURA       CL
1216  2002   ACURA       EL
```

index

```
cars_grouped_year = df[['make','model']].groupby(df['year'])
for year, group in cars_grouped_year:
    print(year)
    print(group)   # dataframe - make and model
```

group

year *df[['make','model']]*

```
2001
         make     model
0       ACURA        CL
1       ACURA        EL
...       ...       ...
1213   YAMAHA    YZF-R6
1214   YAMAHA   YZF600R

[1215 rows x 2 columns]
2002
         make     model
1215    ACURA        CL
1216    ACURA        EL
...       ...       ...
```

index

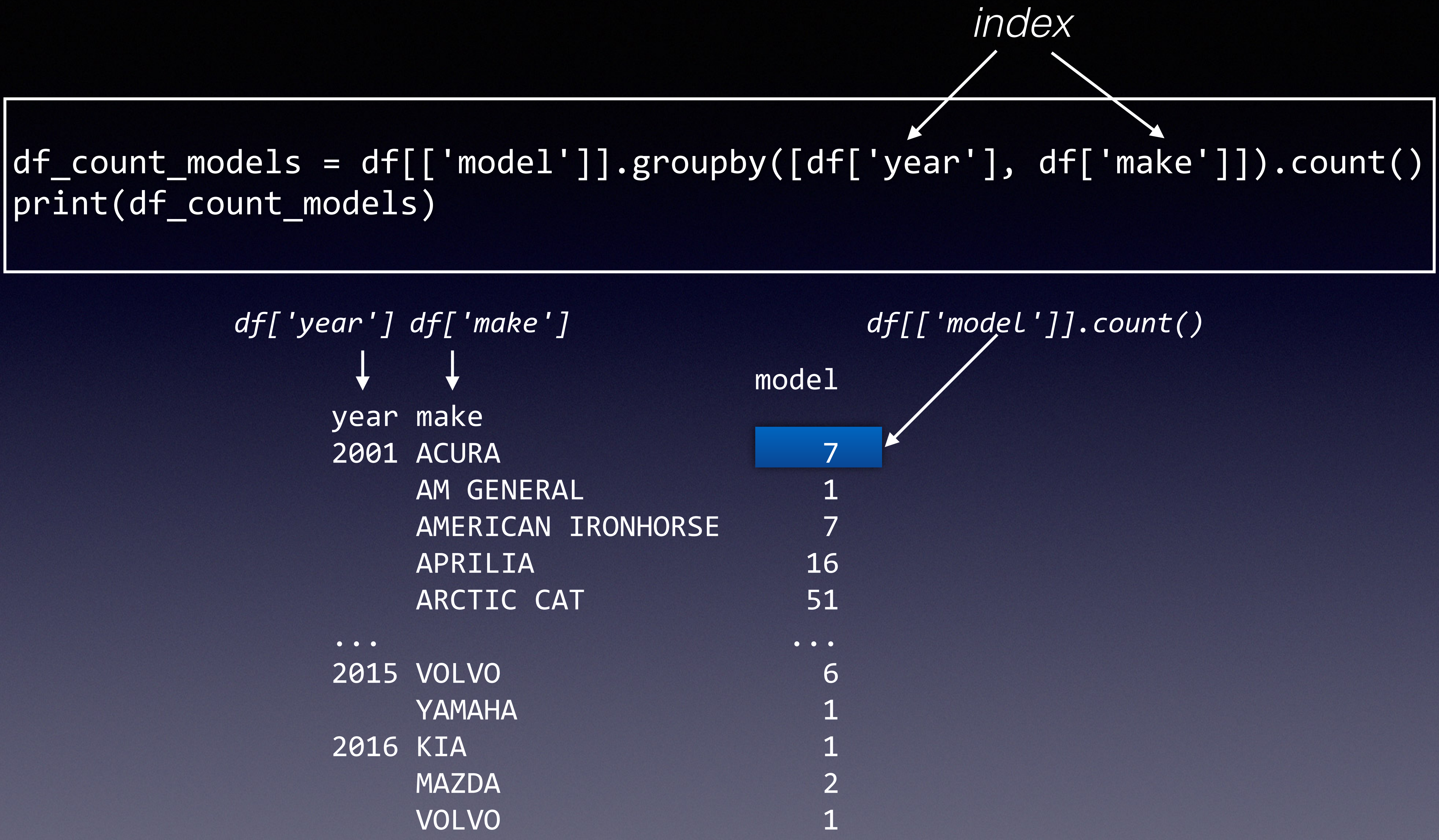
```
cars_grouped_year_make = df[['model']].groupby([df['year'], df['make']])
for year_make, group in cars_grouped_year_make:
    print(year_make)
    print(group)     # dataframe - model
    print(f'Sub-total - {group.count()}')
```

year_make

```
(2001, 'ACURA')
     model
0       CL
1       EL
2  INTEGRA
3      MDX
4      NSX
5       RL
6       TL
Sub-total - 7
(2001, 'AM GENERAL')
    model
7  HUMMER
...
```

group

df[['make']]



index

```
df_count_models = df[['model']].groupby([df['year'], df['make']]).count()
print(df_count_models)
```

df['year'] df['make']

df[['model']].count()

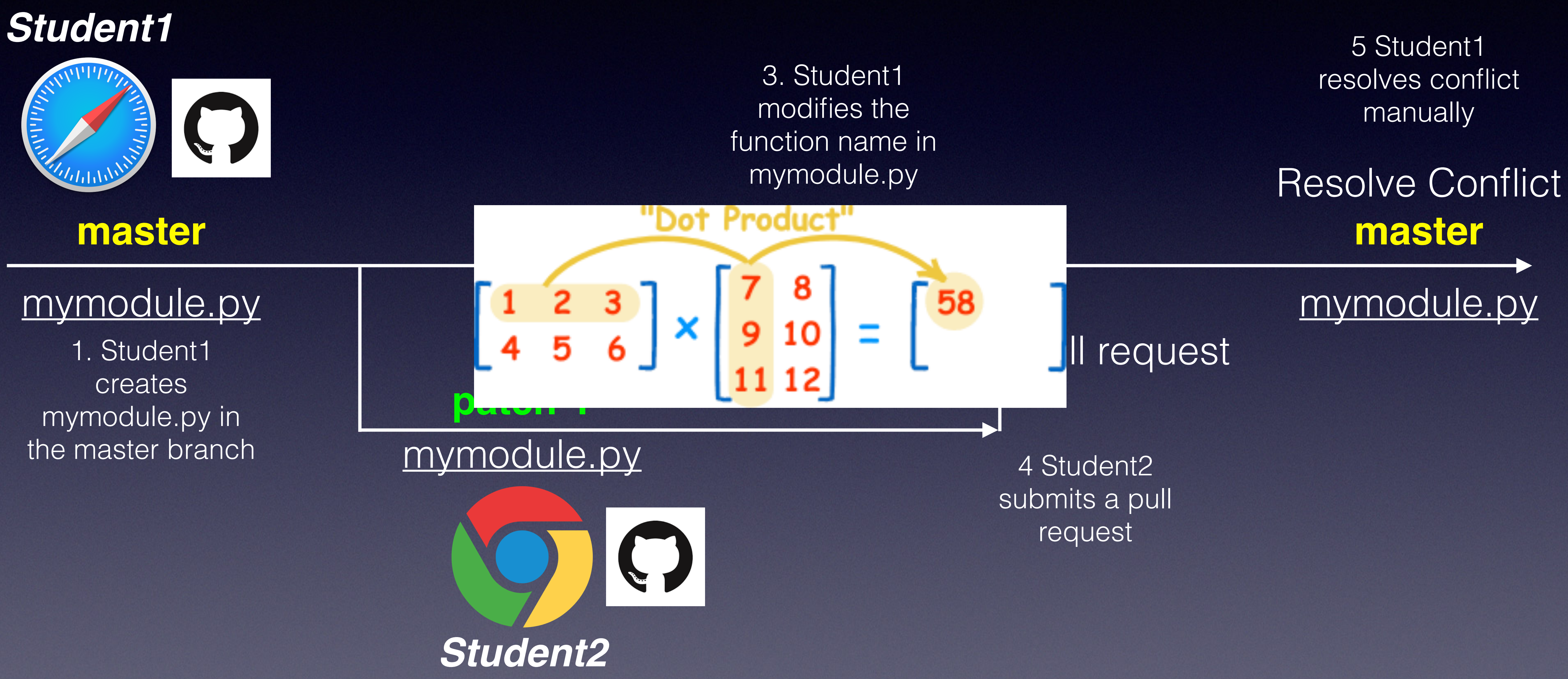
```
                             model
year make
2001 ACURA                       7
     AM GENERAL                  1
     AMERICAN IRONHORSE          7
     APRILIA                    16
     ARCTIC CAT                 51
...                            ...
2015 VOLVO                       6
     YAMAHA                      1
2016 KIA                         1
     MAZDA                       2
     VOLVO                       1
```

GitHub

GitHub

- **Git** is an open-source version control system that was started by Linus Torvalds—the same person who created Linux.
- **GitHub** - where developers store their projects and network with like minded people.

Resolving Conflicts



mymodule.py

1. Student1 creates mymodule.py in the master branch

mymodule.py

2. Student2 modifies the function name in mymodule.py

3. Student1 modifies the function name in mymodule.py

4 Student2 submits a pull request

5 Student1 resolves conflict manually

mymodule.py