Numerical Python

NumPy
Roadmap
requests Redux
import requests
page = requests.get( 'mydataurl.com/data.csv' )
data = ???

This code should produce a list containing the comma-separated numbers at the URL. What should replace the ????

A text.split(',
B page.text.split(',
C text().split(',
D page.text().split(',

requests Redux
import requests
page = requests.get('mydataurl.com/data.csv')
data = page.text.split(',,')

This code should produce a list containing the comma-separated numbers at the URL. What should replace the `????`?

A. `text.split(',,')`
B. `page.text.split(',,')` ⭐
C. `text().split(',,')`
D. `page.text().split(',,')`
x = 'ABBACAB'
d = {}
for c in x:
    if c not in d:
        d[ c ] = 0
    d[ c ] += 1

What is the final value of d?
A { 'A': 3, 'C': 1, 'B': 3 }
B { 'A': 3, 'B': 3, 'C': 1 }
C { 'AAA', 'BBB', 'C' }
D [ 3, 3, 1 ]
x = 'ABBACAB'
d = {}
for c in 'ABBACAB':  #c = 'A', 'B', etc.
    if c not in d:
        d[ c ] = 0
    d[ c ] += 1

c='A' => d['A']=0
c='A' => d['A']+=1
c='B' => d['B']=0
c='B' => d['B']+=1
c='B' => d['B']+=1
...

x = 'ABBACAB'
d = {}
for c in x:
    if c not in d:
        d[ c ] = 0
        d[ c ] += 1

What is the final value of d?

A  { 'A': 3, 'C': 1, 'B': 3 }  ✴
B  { 'A': 3, 'B': 3, 'C': 1 }  ✴
C  { 'AAA', 'BBB', 'C' }
D  [ 3, 3, 1 ]

Why both?
d = {}
for n in d:
    print( n )

What does this code print?
A  The values of d.
B  The keys of n.
C  The key–value pairs of d.
d = { 'red':1, 'green':2, 'blue':3 }
for n in d:
    print( n )

red
blue
green
d = {}
for n in d:
    print( n )

What does this code print?

A  The values of d.
B  The keys of n.
C  The key–value pairs of d.
d = {}
for n in d:
    print(n)

What does this code print?

A The values of d.
B The keys of n. ✴
C The key–value pairs of d.

▷ So how do you access value corresponding to key n?
d = {}
for n in d:
    print(n)

What does this code print?
A  The values of d.
B  The keys of n. ✰
C  The key–value pairs of d.

So how do you access value corresponding to key n?
-du d[n]
zipcode = { 'Bill': 60644, 'Jill': 41073, 'Tony': 63103 }
city = { 60644: 'Chicago', 41073: 'Cincinnati', 63103: 'St. Louis' }
x = city[ zipcode[ 'Tony' ] ]

What is the final value of x?
A '63103'
B 'St. Louis'
C 'Chicago'
D None
zipcode = { 'Bill': 60644,  
            'Jill': 41073,  
            'Tony': 63103 }  

city = { 60644: 'Chicago',  
         41073: 'Cincinnati',  
         63103: 'St. Louis' }  

x = city[ zipcode[ 'Tony' ] ]

What is the final value of x?

A '63103'
B 'St. Louis' ★
C 'Chicago'
D None
Arrays
mydata = [4.5, 6.0, 1.2, 5.4]
from math import sin
sin(mydata)

- Why doesn’t this work?
- list can contain any type!
- Also operators don’t do what we “want”:
  mydata * 2.0  # doesn’t double values!
import numpy
import numpy as np  # rename it, it’s easier
numpy

import numpy
import numpy as np  # rename it, it's easier

- numpy provides arrays and mathematical functions.

data = np.array([ 4.5, 6.0, 1.2, 5.4])
data * 2.0
numpy

```python
x = np.array([[1,2], [3,4]])
np.zeros((3,3))
np.ones((4,4))
np.eye(4)
x.shape
x.dtype
x = np.array([[1,2], [3,4]], dtype=np.float64)
```
x[:,1]
x[1,:]
x.T
x.tolist()
x.sort()
x.argsort()
x * x  # element-wise, not matrix-like!
Consider a data set containing patient inflammation records for 60 patients over a period of 40 days, contained in inflammation.csv.

```python
data = np.loadtxt( './data/inflammation.csv', delimiter=',' )
print( data.shape )
```
numpy

Arrays can be a bit tricky to keep straight; test them if you need to.

Axes
import numpy as np
x = np.array( [ 5,1,3 ] )
x *= 2

What is the value of x?

A [ 10,2,6 ]
B array( [ 10,2,6 ] )
C [ 5,1,3,5,1,3 ]
D array( [ [ 5,1,3 ], [ 5,1,3 ] ] )
import numpy as np
x = np.array([5, 1, 3])
x *= 2

What is the value of x?

A [ 10, 2, 6 ]
B ★
  array([10, 2, 6])
C [ 5, 1, 3, 5, 1, 3 ]
D array([[ 5, 1, 3 ], [ 5, 1, 3 ]])
import numpy as np
x = np.array([1] * 2)
x += 1

What is the final value of x?

A  array([2])
B  array([1,1,1])
C  array([2,2])
D  array([3])
import numpy as np
x = np.array([1] * 2)
x += 1

What is the final value of x?

A array([2])
B array([1, 1, 1])
C array([2, 2])
D array([3])
Indexing arrays

numpy indexes by `array[row][col]` or `array[row, col]`.
numpy supports many possible data types:
- bool
- int16, int32, int64
- float16, float32, float64
- complex64, complex128

You frequently don’t need to specify the type.
For the most part, stick with bool, int64, and float64 (most accurate).

Specify (and query) with dtype method:

```python
a = [ 3,2,4 ]
x = np.array( a,dtype=np.float64 )
x.dtype
```
Other arrays

```python
x = np.zeros([2,3])  # zeroes
y = np.ones([4,1])   # ones
```

- Produce arrays of zeros or ones with specified dimensions.
z = np.eye( 5 ) # identity

- Produces identity matrix of specified square dimension.
Other arrays

\[ w = \text{np.linspace}(0,10,101) \]
\[ v = \text{np.linspace}(\text{start}, \text{finish}, n) \]

- Produce arrays from \text{start} to \text{finish} of \text{n} points (\textit{not} spacing!).
- Excellent for grids and coordinates.
- May also see \texttt{arange}, but I recommend avoiding its use:

\[ u = \text{np.arange}(0,10,0.1) \# \text{tricky!} \]
\[ u == \text{array}( [ 0, 0.1, 0.2, \ldots, 9.9 ] ) \]