Dictionaries
Mutability Redux
```python
x = [ 'a', 'b' ]
y = [ 'c', 'd' ]
def add_it( a,b ):
    b.append( a )
add_it( y,x )
```

What is the final value of `x`?

A [ 'a', 'b', 'c', 'd' ]  
B [ 'a', 'b' ]  
C [ 'a', 'b', [ 'c', 'd' ] ]  
D [ 'c', 'd', [ 'a', 'b' ] ]  
E None
x = [ 'a', 'b' ]
y = [ 'c', 'd' ]
def add_it( a,b ):
    b.append( a )
add_it( y,x )

What is the final value of x?
A [ 'a', 'b', 'c', 'd' ]
B [ 'a', 'b' ]
C [ 'a', 'b', [ 'c', 'd' ] ] *
D [ 'c', 'd', [ 'a', 'b' ] ]
E None
```python
x = ['a', 'b']
y = ['c', 'd']
def add_it( a,b):
    b.append( a )
add_it( y,x )
```

What is the final value of `x`?

A [ 'a', 'b', 'c', 'd' ]
B [ 'a', 'b' ]
C [ 'a', 'b', [ 'c', 'd' ] ]
D [ 'c', 'd', [ 'a', 'b' ] ]
E None
x = ['a', 'b', 'c', 'd']
y = x.sort()
z = (y is x)

What is the final value of z?
A True
B False
Be sure to distinguish:

```python
x = ['a','b','c','d']
y = x
z = x[ : ]

# The following are all True:
x is y
z is not x
x == y
x == z
```
Multidimensional Indexing
Just as we can nest control structures, we can nest container values.

\[ a = \left[ \left[ 1, 2 \right], \left[ 3, 4 \right] \right] \]

What does this look like to you?
Nested lists

Just as we can nest control structures, we can nest container values.

\[ a = \begin{bmatrix} [1, 2], [3, 4] \end{bmatrix} \]

What does this look like to you? A matrix.
Access member values of a nested container by coordinates:

```python
a = [ [ 1, 2 ], [ 3, 4 ] ]
a[0]  #?
a[0][0]  #?
```

Python orders by \((row, column)\) — that is, the first number selects the row and the second selects the column in that row.
Multidimensional indexing

- Access member values of a nested container by coordinates:
  
  \[
  a = \begin{bmatrix}
  1, 2 \\
  3, 4 
  \end{bmatrix}
  \]

  \[
  a[0] \\
  a[0][0] \\
  \]

- Python orders by \((row, column)\)—that is, the first number selects the row and the second selects the column in that row.

- Side effect: easy to select “rows”, hard to select “columns”!
Example

$$a = \begin{bmatrix} [1,2,3], [4,5,6], [7,8,9] \end{bmatrix}$$

How would you refer to the value 6?

A  $a[2][3]$
B  $a[1][2]$
C  $a[2,3]$
D  $a[2][1]$
Example

Multidimensional Indexing

$$a = \begin{bmatrix} [1,2,3], & [4,5,6], & [7,8,9] \end{bmatrix}$$

How would you refer to the value 6?

A. \(a[2][3]\)
B. \(a[1][2]\) ★
C. \(a[2,3]\)
D. \(a[2][1]\)
Dictionaries
How do we index an element of a list?
How do we index an element of a list?
- lists and tuples are *ordered*, so ints make sense.
- What else could work to look up data?
list data type

Dictionaries
Dictionaries
model = {
    'Civic': 'Honda',
    'Mustang': 'Ford',
    'Model S': 'Tesla',
    'Model T': 'Ford'
}
The `dict` indexes data by *any* value (*unordered*).

Easy to think of as dictionary, but can use lots besides strings.

This container maps *keys* to *values*.

```
key       value
''911''   'Porsche'
cars['911'] = 'Porsche'
```
dict data type

cars = {}
cars[ 'Mustang' ] = 'Ford'
cars[ 'Viper' ]   = 'Dodge'
cars[ 'Corvette' ]= 'Chevrolet'
cars[ 'Charger' ] = 'Dodge'
cars[ '911' ]     = 'Porsche'
We create a `dict` as follows:
- opening brace `{`
- key : value pairs, separated by commas
- closing brace `}`

Keys can be any *immutable* type: `int`, `float`, `str`, `tuple`
Dictionaries

d = { 'one':1, 'two':2, 'three':3, 4.0:4 }
print( d['one'] )
d[ 5 ] = 'five'  # map int 5 to str 'five'
del d[ 4.0 ]
'six' in d
for key in d:    # no guarantee on order
    print( key, d[key] )
d.keys()
d.values()
Example

\[ d = \{ \text{'a':2, 'c':3, 'b':1} \} \]
\[ x = d[\text{'a'}] + d[\text{'c'}] \]

What is the final value of \(x\)?

A. 4
B. 'ac'
C. '5'
D. 5
Example

d = { 'a':2, 'c':3, 'b':1 }
x = d[ 'a' ] + d[ 'c' ]

What is the final value of x?
A 4
B 'ac'
C '5'
D 5
d = { 'a': '2', 'c': '3', 'b': '1' }

x = d[ 'a' ] + d[ 'c' ]

What is the final value of x?

A  23
B  5
C  '23'
D  '5'
Example

d = {'a':'2', 'c':'3', 'b':'1'}
x = d['a'] + d['c']

What is the final value of x?
A 23
B 5
C '23' ⭐
D '5'
d = { }
words = [ 'red', 'orange', 'yellow' ]
for word in words:
    d[ word ] = words.index( word )

What is the final value of d?
A { 'red':3, 'orange':6, 'yellow':6 }
B { 'red':0, 'orange':2, 'yellow':2 }
C None
D { 'orange': 1, 'red': 0, 'yellow': 2 }
E { 'red': 0, 'orange': 1, 'yellow': 2 }
d = { }
words = [ 'red', 'orange', 'yellow' ]
for word in words:
    d[ word ] = words.index( word )

What is the final value of d?
A { 'red':3, 'orange':6, 'yellow':6 }  
B { 'red':0, 'orange':2, 'yellow':2 }  
C None  
D { 'orange': 1, 'red': 0, 'yellow': 2 }  
E { 'red': 0, 'orange': 1, 'yellow': 2 }
Dictionaries can encode/decode data, or translate from one representation to another.
Dictionaries can encode/decode data, or translate from one representation to another.

```python
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
e = {}
for i in range(len(x)):
    e[x[i]] = y[i]
encoded = ''
for c in 'HELLO':
    encoded += e[c]
```

How would you reverse (decode) this?
x = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
y = 'BCDEFGHIJKLMNOPQRSTUVWXYZA'
d = {}
for i in range(len(x)):
    d[y[i]] = x[i]
decoded = ''
for c in encoded:
    decoded += d[c]
Exercise

- Encode all of the words in a string using a Caesar cipher.
- Decode all of the words in the string.
# e,d from before
s = 'flannel vest'
s = s.upper()
encoded = ''
for c in s:
    if c not in e:
        encoded += c
        continue
    encoded += e[c]
# e,d,encoded from before
decoded = ''
for c in encoded:
    if c not in d:
        decoded += c
    continue
decoded += e[c]
Dictionaries can also function as accumulators.

```python
# count the vowels
vowels = 'aeiou'
text = 'In Xanadu did Kubla Khan/A stately pleasure-dome decree'

text = text.lower()
d = { 'a':0,'e':0,'i':0,'o':0,'u':0 }
for c in text:
    if c in vowels:
        d[ c ] += 1
```
Count the category frequencies in Jeopardy questions.
We can link data based on a common field.

```python
dictionary = { 'Bill': 60644,
              'Jill': 41073,
              'Tony': 63103 }

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

def print_name_city(name, city):
    print(f'{name}: {city[zipcode[name]]}')

for name in zipcode:
    print_name_city(name, city)
```