Dictionary Comprehensions

WE'LL COVER THE FOLLOWING

A dictionary comprehension is like a list comprehension, but it constructs a dictionary instead of a list.

```
import os, glob

metadata = [(f, os.stat(f)) for f in glob.glob('*test*.py')]            #①
print (metadata[0] )                                                    #②
# ('romantest3.py', os.stat_result(st_mode=33261,
# st_ino=1057919, st_dev=2049, st_nlink=1, st_uid=1003,
# st_gid=50, st_size=4640, st_atime=1472210494,
# st_mtime=1472151083, st_ctime=1472210494))

metadata_dict = {f:os.stat(f) for f in glob.glob('*test*.py')}          #③
print (type(metadata_dict))                                             #④
#<class 'dict'>

print (list(metadata_dict.keys()))                                      #⑤
#['romantest2.py', 'romantest3.py', 'romantest1.py']

print (metadata_dict['romantest3.py'].st_size)                          #⑥
#4640
```

① This is not a dictionary comprehension; it's a list comprehension. It finds all `.py` files with `test` in their name, then constructs a tuple of the filename and the file metadata (from calling the `os.stat()` function).

② Each item of the resulting list is a tuple.

③ This is a dictionary comprehension. The syntax is similar to a list comprehension, with two differences. First, it is enclosed in curly braces instead of square brackets. Second, instead of a single expression for each

item, it contains two expressions separated by a colon. The expression before

the colon (f in this example) is the dictionary key; the expression after the colon (`os.stat(f)` in this example) is the value.

④ A dictionary comprehension returns a dictionary.

⑤ The keys of this particular dictionary are simply the filenames returned from the call to `glob.glob('*test*.py')`.

⑥ The value associated with each key is the return value from the `os.stat()` function. That means we can "look up" a file by name in this dictionary to get its file metadata. One of the pieces of metadata is `st_size`, the file size. The file `romantest2.py` is `4640` bytes long.

Like list comprehensions, you can include an `if` clause in a dictionary comprehension to filter the input sequence based on an expression which is evaluated with each item.

```
import os, glob, humansize

metadata_dict = {f:os.stat(f) for f in glob.glob('*')}                          #①
humansize_dict = {os.path.splitext(f)[0]:humansize.approximate_size(meta.st_size)
                  for f, meta in metadata_dict.items() if meta.st_size > 6000}  #②

print (list(humansize_dict.keys()))                                             #③
#['data.tar']

print (humansize_dict['data.tar'])                                              #④
#7.2 KiB
```

① This dictionary comprehension constructs a list of all the files in the current working directory (`glob.glob('*')`), gets the file metadata for each file (`os.stat(f)`), and constructs a dictionary whose keys are filenames and whose values are the metadata for each file.

② This dictionary comprehension builds on the previous comprehension, filters out files smaller than `6000` bytes (if `meta.st_size > 6000`), and uses that filtered list to construct a dictionary whose keys are the filename minus the extension (`os.path.splitext(f)[0]`) and whose values are the approximate size of each file (`humansize.approximate_size(meta.st_size)`).

③ As you saw in a previous example, there are six such files, thus there are six items in this dictionary.

④ The value of each key is the string returned from the `approximate_size()` function.

Other Fun Stuff To Do With Dictionary Comprehensions

Here's a trick with dictionary comprehensions that might be useful someday: swapping the keys and values of a dictionary.

```
a_dict = {'a': 1, 'b': 2, 'c': 3}
print ({value:key for key, value in a_dict.items()})
#{1: 'a', 2: 'b', 3: 'c'}
```

Of course, this only works if the values of the dictionary are immutable, like strings or tuples. If you try this with a dictionary that contains lists, it will fail most spectacularly.

```
a_dict = {'a': [1, 2, 3], 'b': 4, 'c': 5}
print ({value:key for key, value in a_dict.items()})
#Traceback (most recent call last):
# File "/usercode/__ed_file.py", line 2, in <module>
# print ({value:key for key, value in a_dict.items()})
# File "/usercode/__ed_file.py", line 2, in <dictcomp>
# print ({value:key for key, value in a_dict.items()})
#TypeError: unhashable type: 'list'
```