# Dictionaries and Lists

Isaac Dudney, Jeremy Hansen, Aida Popa

We'll be comparing and showing runtimes of list and dictionary functions. It'll be split into 3 parts: dictionaries, lists, and the common functions between them. The dictionary and list sections will simply report runtimes of functions on various inputs. The common functions will compare the two.

## Lists

## Dictionaries

Dictionaries have 4 functions that are not shared with lists.

At 200,000 key/value pairs, the slowest operation was grabbing the items (`items`) at 0.31 seconds, followed by grabbing all values (`values`), which took 0.21 seconds. Grabbing just the keys (`keys`), however, took only 0.069 seconds, which is an order of magnitude faster than either other operation; it seems grabbing keys is much easier than grabbing their values, suggesting that the only way to grab a value is through its key (or else accessing keys and values should take around the same time). The fastest function was the `get` function, at only 0.052 seconds, faster by a small margin than grabbing all keys. The `get` function provides a specific key/value pair, accessed by key.

## Common

Lists and dictionaries compared is interesting, as each operation in common has some differences that indicate usefulness in different areas. All times were measured on an i7-4600U running the stock Ubuntu Linux kernel and using Python 3.5.2. The process for measuring differences was simple: create a function or function pair that would test analogous functions 10,000 times and average the running time for each. The output was then printed; the code is in `common.py`. Generating a list and a dictionary of 100 items each took almost exactly the same amount of time, at 110.5 microseconds for the list, and 109.6 for the dictionary. This can be expected, as the code for each generation is as close as possible: it generates an equal number of random numbers to fill the values and items (of dictionaries and lists, respectively).

Next, access times. The access function accesses a random item and value by generating a random number bounded by the lengths of the dictionary and list, and returns the amount of time it took to access it. The dictionary was faster by a slim margin, at only 98 nanoseconds, and lists at 105 nanoseconds. This variation is incredibly slim, but the running time is averaged over 100,000 runs; that should smooth out a majority of the variations in running time, but at such a close call, we can relatively safely say they're about even in access time.

Next, checking if an item exists in a collection. Interestingly, dictionaries took about the same amount of time to check regardless of if the item actually existed or not; 121 nanoseconds if the item existed, and 108 nanoseconds if it didn't. Lists were considerably slower, at 596 nanoseconds if it existed, and 1,132 if it didn't. This indicates that if you need to check for an item's existence in a structure, lists are much faster. Constant checks like a search would almost certainly be faster using a dictionary.

Appending items is the next thing to test. We appended 10,000 items to each collection. Lists were slower, at 146 nanoseconds, and 134 nanoseconds for dictionaries. This, again, is not entirely unexpected; the differences between generation were slim, but they only generated 100 items vs the 10,000 items appended here. The time measured in the generation is the generation of the entire collection, while the time measured here is the average time per append. So, the differences between the two may be magnified here. Overall, dictionaries are slightly faster at appending and generation.

Finding the length of a collection is relatively uninteresting, both taking 130 nanoseconds, suggesting the length code is standardized across collections.

Deleting an item from a list, however, takes considerably longer than deleting a pair from a dictionary. Lists on average take 3,373 nanoseconds to remove an item. Dictionaries take only 339 nanoseconds. This is a factor of ten off, indicating that dictionaries have a much easier time deleting items. It's possible that lists have to perform more operations on a deletion, while dictionaries only have to dereference one hash for the structure to stay the same. The list may have an index that it has to update along with clearing any item's reference to the deleted item, leading to the increased runtime.

Overall, dictionaries seem like faster constructs than lists in general; perhaps there is some other metric we are missing, like iteration time (as discussed in the 1st draft), or memory usage, but other than the first one, it is outside the scope of this assignment.