LECTURE 10

DICTIONARIES

MCS 260 Fall 2020

Emily Dumas

REMINDERS

- Work on:
  - Project 1 (due Friday, 6pm central)
  - Worksheet 4
  - Quiz 4
- Docstrings now required for all functions & files
- Come to Tue/Thu discussion ready to share screen and code (if at all possible)

DICTIONARIES

Lists and tuples are sequences: they store an ordered collection of values that can be retrieved by index (a nonnegative integer).

A **dictionary** or **dict** in Python is an *unordered* collection of *key:value* pairs. Values can be retrieved using the associated key, similar to indexing a list.

The values in a dictionary can be of any type, but there are some restrictions on the keys.

Dictionaries are mutable.

Example of syntax for working with dictionaries:

```
>>> # define a new dict
>>> tuition = { "UIC": 10584,
...             "Stanford": 50703,
...             "Harvard": 46340 }
>>> # Access an item
>>> tuition["UIC"]
10584
>>> # Add or change an item
>>> tuition["PSU"] = 18454
>>> tuition
{'UIC': 10584,
 'Stanford': 50703,
 'Harvard': 46340,
 'PSU': 18454}
>>> # Remove an item
>>> del tuition["Harvard"]
>>> tuition
{'UIC': 10584, 'Stanford': 50703, 'PSU': 18454}
```

Mixed types are ok for keys or values.

```
d = { 1: "fish", "two": "fish", "x": [7,6,5] }
```

Methods:

```
>>> d.keys()   # All keys (like range(len(L)))
dict_keys([1, 'two', 'x'])
>>> d.items()  # All key-value pairs (like enumerate(L))
dict_items([(1, 'fish'), ('two', 'fish'), ('x', [7, 6, 5])])
>>> d.values() # All values
dict_values(['fish', 'fish', [7, 6, 5]])
```

dict_keys, dict_items, dict_values types behave a lot like list, and can be converted to a list with `list()`.

MEMBERSHIP TESTING

Membership in a dictionary means being a *key*!

```
>>> d = { 1: "fish", "two": "fish", "x": [7,6,5] }
>>> "fish" in d
False
>>> 1 in d
True
```

Forgetting this is a very common source of programming errors.

OTHER LANGUAGES

Python dicts are examples of **associative arrays**, also known as **maps**.

In other languages with a built-in associative array type, the type is often called *map* or *Map* (e.g. in C++, Java, Go)

The rules (allowable keys, type heterogeneity, etc.) vary by language.

ITERATION OVER DICTS

dicts are iterable, but iterate over the *keys*.

```
for k in d:  # loop over keys
    print(k,"is one of the keys")

for k in d:  # loop over keys (index to get value)
    print("Key",k,"has value",d[k])

for k,v in d.items(): # loop over keys,value pairs
    print("Key",k,"has value",v)
```

It is common for the values in a dict to be dicts themselves. This is the usual way to make a collection of labeled data indexed by a key.

```
schooldata = {
    "UIC": {
        "fullname": "University of Illinois at Chicago",
        "tuition": 10584,
        "undergrad_students": 21641,
        },
    "Stanford": {
        "fullname": "Leland Stanford Junior University",
        "tuition": 50703,
        "undergrad_students": 7083
        },
    "Harvard": {
        "fullname": "Harvard University",
        "tuition": 46340,
        "undergrad_students": 6755
```

DICTIONARIES AS RULES

```
pr_replacements = {
    "accident": "unplanned event",
    "escape": "departure",
    "laser-sharks": "fish"
}
original = "an accident involving the escape of laser-sharks"
words = original.split()  # [ "an", "accident", ... ]
for w in words:
    if w in pr_replacements:
        w = pr_replacements[w]
    print(w,end=" ")
print()
```

Output:

```
an unplanned event involving the departure of fish
```

HASHABLE TYPES

Not all types in Python can be used as dict keys.

```
>>> d = dict() # empty dict
>>> d[ [3,4,5] ] = 6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> d[ { 5:"five" } ] = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'dict'
```

Keys must allow *hashing* which typically requires immutability.

Strings, tuples, and numeric types are all hashable. Lists and dicts are not.

You can check if a value is hashable using the built-in **`hash()`** function:

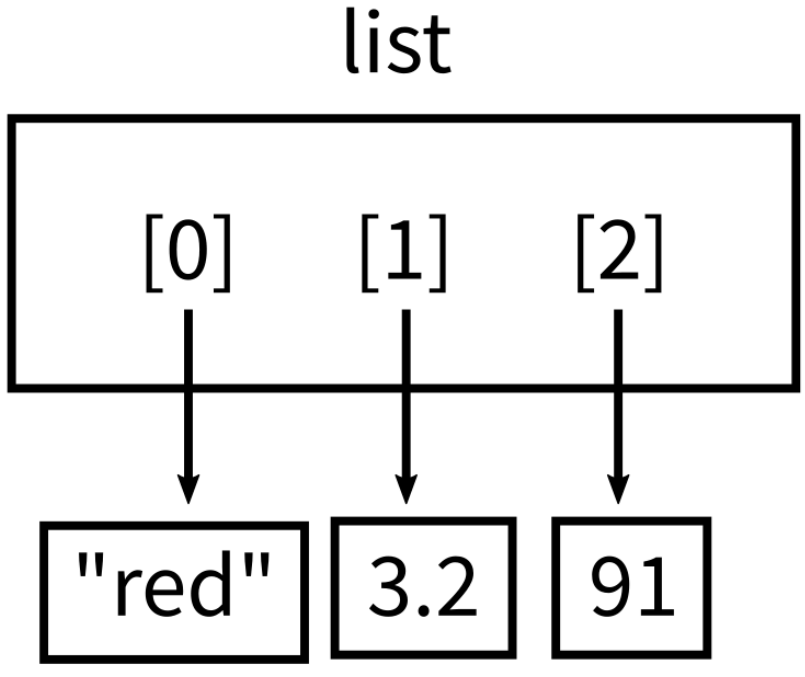
```
>>> hash(1)
1
>>> hash(1.5)
1152921504606846977
>>> hash("Granny Smith")
2634656644181978377
>>> hash( [1,2,3] )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

DICTIONARY COMPREHENSIONS

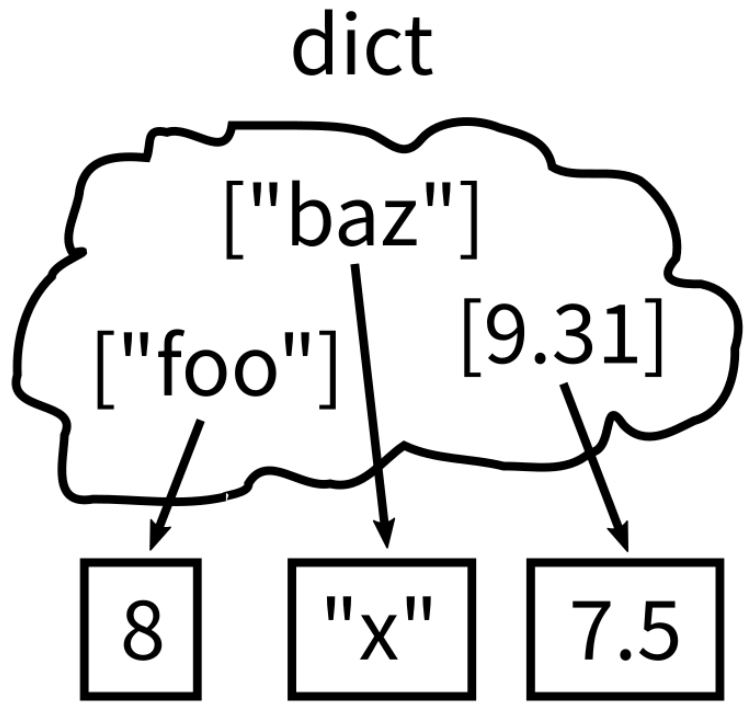
Analogous to list comprehensions, but using

`{ key:value for name in iterable ... }`

```
>>> words = [ "Chicago", "cat", "cinemas" ]
>>> word_data = { w: { "length": len(w),
...                    "all lower": w==w.lower() }
...                for w in words }
>>> word_data
{'Chicago': {'length': 7, 'all lower': False},
 'cat': {'length': 3, 'all lower': True},
 'cinemas': {'length': 7, 'all lower': True}
}
```



index to value
ordered



key to value
unordered

REFERENCES

- In *Downey*:
  - Chapter 11 covers dictionaries

ACKNOWLEDGEMENT

- Some of today's lecture was based on teaching materials developed for MCS 260 by Jan Verschelde.

REVISION HISTORY

- 2020-09-15 Initial publication