Key-Value Pairs

- The last built-in type: dictionary (or dict)
  - One of the most important in all of Python
  - Like a list, but built of key-value pairs
- **Keys:** Unique identifiers
  - Think social security number
  - At Cornell we have netids: jrs1
- **Values:** Non-unique Python values
  - John Smith (class '13) is jrs1
  - John Smith (class '16) is jrs2

Idea: Lookup values by keys

1

Basic Syntax

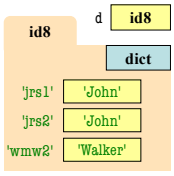
- Create with format: `{k1:v1, k2:v2, ...}`
  - Both keys and values must exist
  - **Ex:** `d={'jrs1':'John','jrs2':'John','wmw2':'Walker'}`
- **Keys** must be **non-mutable**
  - ints, floats, bools, strings, tuples
  - **Not** lists or custom objects
  - Changing a key's contents hurts lookup
- **Values** can be **anything**

2

Using Dictionaries (Type dict)

- Access elts. like a list
  - `d['jrs1']` evals to `'John'`
  - `d['jrs2']` does too
  - `d['wmw2']` evals to `'Walker'`
  - `d['abc1']` is an **error**
- Can test if a key exists
  - `'jrs1'` in `d` evals to `True`
  - `'abc1'` in `d` evals to `False`
- But cannot slice ranges!

```
d = {'js1':'John','js2':'John',
     'wmw2':'Walker'}
```



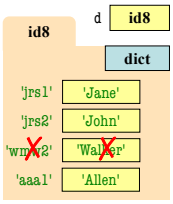
Key-Value order in folder is not important

3

Dictionaries Can be Modified

- Can reassign values
  - `d['jrs1'] = 'Jane'`
  - Very similar to lists
- Can add new keys
  - `d['aaa1'] = 'Allen'`
  - Do not think of order
- Can delete keys
  - `del d['wmw2']`
  - Deletes both key, value

```
d = {'jrs1':'John','jrs2':'John',
     'wmw2':'Walker'}
```



4

Dictionaries: Iterable, but not Sliceable

- Can loop over a dict
  - Only gives you the keys
  - Use key to access value

```
for k in d:
    # Loops over keys
    print(k)     # key
    print(d[k])  # value
```

- Can iterate over values
  - **Method:** d.values()
  - But no way to get key
  - Values are not unique

```
# To loop over values only
for v in d.values():
    print(v)     # value
```

5

Dictionary Loop with Accumulator

```
def max_grade(grades):
    """Returns max grade in the grade dictionary
    Precondition: grades has netids as keys, ints as values"""
    maximum = 0                # Accumulator
    # Loop over keys
    for k in grades:
        if grades[k] > maximum:
            maximum = grades[k]

    return maximum
```

6

Dictionaries and Mutable Functions

- Restrictions are different than list
  - Okay to loop over dictionary to change
  - You are looping over *keys*, not *values*
  - Like looping over positions
- But you **may not add or remove** keys!
  - Any attempt to do this will fail
  - Have to create a key list if you want this

7

But This is Okay

```
def add_bonus(grades,bonus):
    """Gives bonus points to everyone in grades

    Precondition: grades has netids as keys, ints as values.
    bonus is an int."""
    # No accumulator.  This is a procedure

    for student in grades:
        # Modifies the dictionary, but does not change keys
        grades[student] = grades[student]+bonus
```

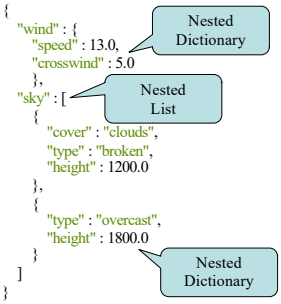
8

Nesting Dictionaries

- Remember, values can be anything
  - Only restrictions are on the keys
- Values can be lists (**Visualizer**)
  - `d = {'a':[1,2], 'b':[3,4]}`
- Values can be other dicts (**Visualizer**)
  - `d = {'a':{'c':1,'d':2}, 'b':{'e':3,'f':4}}`
- Access rules similar to nested lists
  - **Example:** `d['a']['d'] = 10`

9

Example: JSON File



- **JSON:** File w/ Python dict
  - Actually, minor differences
- `weather.json`:
  - Weather measurements at Ithaca Airport (2017)
  - **Keys**: Times (Each hour)
  - **Values**: Weather readings
- This is a *nested* JSON
  - Values are also dictionaries
  - Containing more dictionaries
  - And also containing lists

10

Dictionaries and Recursion

- Dictionaries are **not sliceable**
  - Makes it difficult to do divide and conquer
  - So rare to be used in recursion by itself
  - Often the *answer* to a recursion, not the *input*
- However, the **key list** is sliceable
  - Can recurse on key list, not the dict
  - This requires a helper function
  - Helper is recursive, not the main function

11

The Recursive Version

```
def max_grade_helper(netids,grades):
    """Returns max grade among given netids

    Precond: netids a list of keys in grades, grades a dict w/ int values"""
    # Process small data
    if len(netids) <= 1:
        return grades[netids[0]] if len(netids) == 1 else 0

    # Break it up into left and right
    left  = grades[netids[0]]
    right = max_grade_helper(netids[1:],grades)

    # Combine the answers
    return max(left,right)
```

12