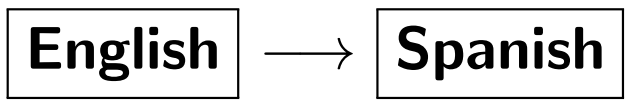
Think Python 2e, Chapter 11 Notes

Dictionaries

September 29, 2022

A dictionary is a **mapping**

English	**Spanish**
one	uno
two	dos
three	tres
cat	gato
dog	perro
priest	sacerdote
...	...



Dictionaries in python

```
>>> eng2sp = dict()
>>> eng2sp
{}
>>> eng2sp['one'] = 'uno'
>>> eng2sp
{'one': 'uno'}
```

The output format is also an input format:

```
>>> eng2sp ={'one':'uno', 'two':'dos', 'three':'tres'}
>>> eng2sp
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

Note that the order of items is random.

Accessing dictionaries in python

```
>>> eng2sp['two']
'dos'
>>> eng2sp['four']
KeyError: 'four'
>>> len(eng2sp)
3
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
>>> vals = eng2sp.values()
>>> 'uno' in vals
True
```

Histograms with dictionaries

```
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d
```

```
>>> h = histogram('brontosaurus')
>>> h
{'a': 1, 'b': 1, 'o': 2, 'n': 1, 's': 2, 'r': 2, 'u':
    2, 't': 1}
```

See my histogram of *Moby Dick*

Looping with dictionaries

```
def print_hist(h):
    for c in h:
        print(c, h[c])
```

```
>>> h = histogram('parrot')
>>> print_hist(h)
a 1
p 1
r 2
t 1
o 1
```

```
>>> for key in sorted(h):
...     print(key, h[key])
a 1
o 1
p 1
r 2
t 1
```

Reverse lookup

```
def reverse_lookup(d, v):
    for k in d:
        if d[k] == v:
            return k
    raise LookupError()
```

```
>>> h = histogram('parrot')
>>> key = reverse_lookup(h, 2)
>>> key
'r'
```

```
>>> key = reverse_lookup(h, 3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in reverse_lookup
LookupError
```

Raise with message

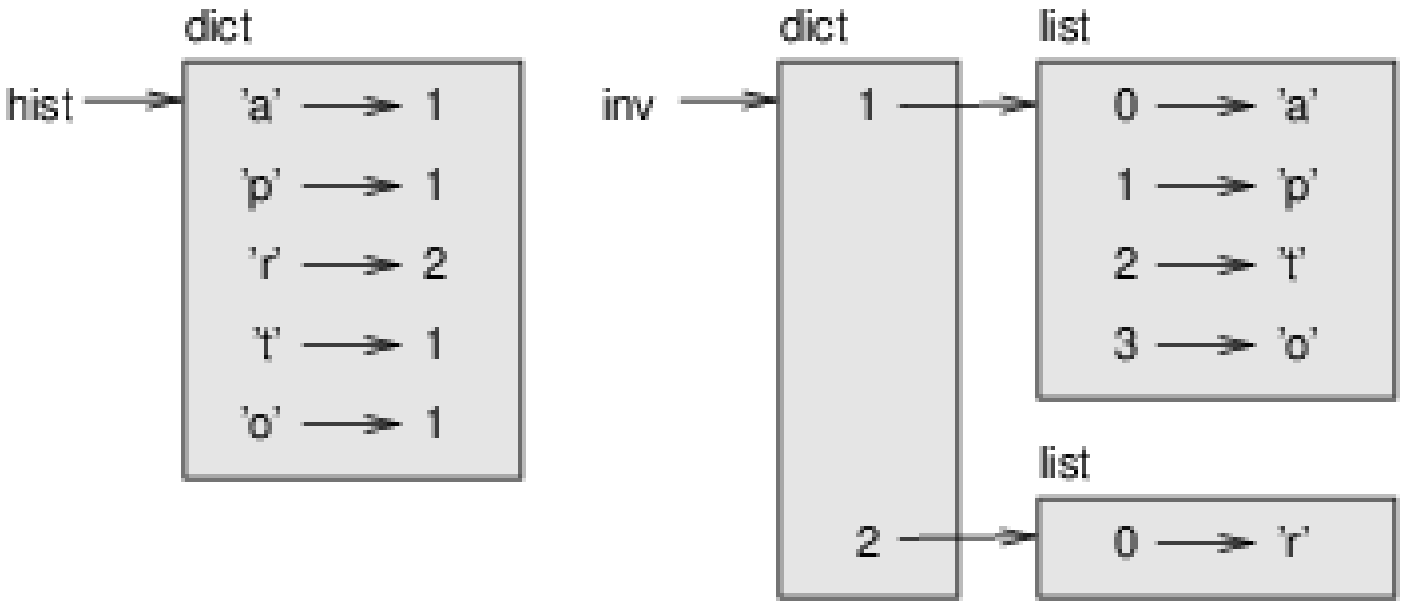
```
>>> raise LookupError('value does not appear in the
    dictionary')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
LookupError: value does not appear in the dictionary
```

Invert a dictionary

```
1 def invert_dict(d):
2     inverse = dict()
3     for key in d:
4         val = d[key]
5         if val not in inverse:
6             inverse[val] = [key]
7         else:
8             inverse[val].append(key)
9     return inverse
```

Invert example

```
>>> hist = histogram('parrot')
>>> hist
{'a': 1, 'p': 1, 'r': 2, 't': 1, 'o': 1}
>>> inverse = invert_dict(hist)
>>> inverse
{1: ['a', 'p', 't', 'o'], 2: ['r']}
```



Lists cannot be keys

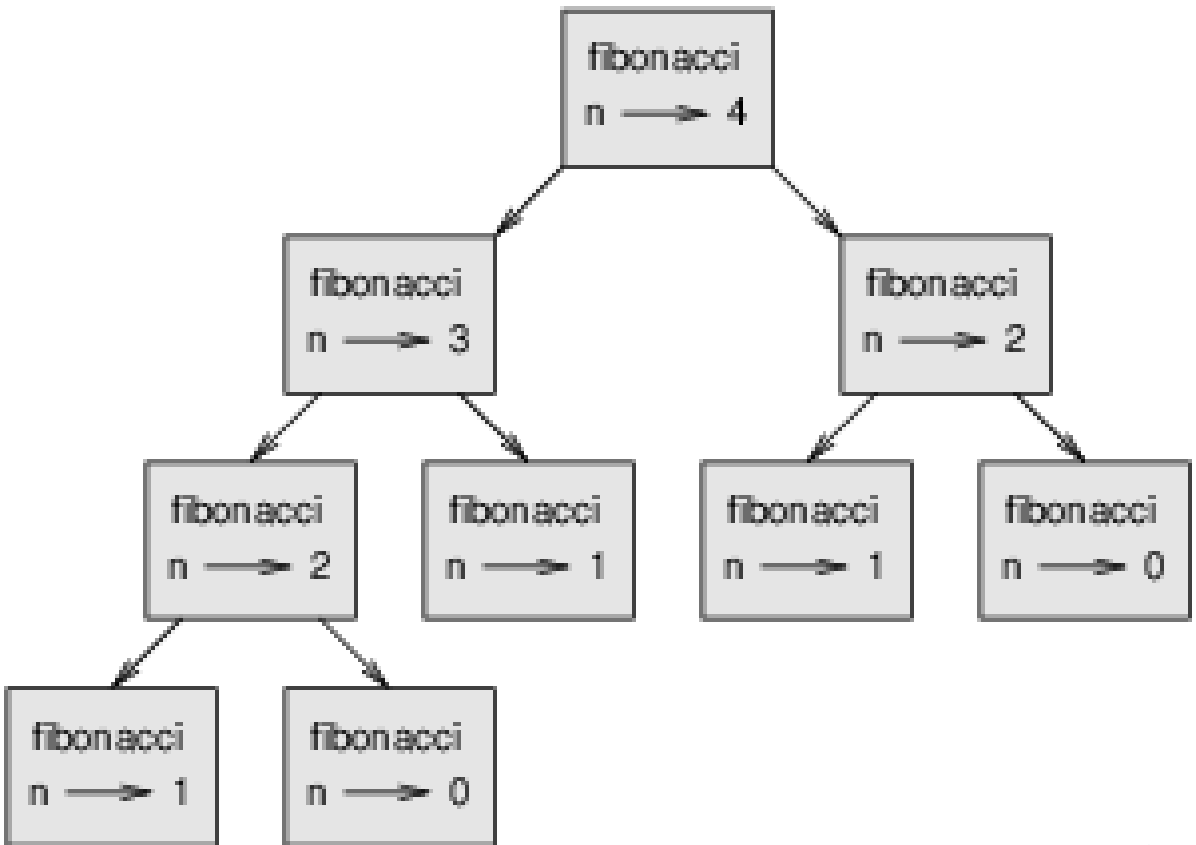
```
>>> t = [1, 2, 3]
>>> d = dict()
>>> d[t] = 'oops'
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: list objects are unhashable
```

Keys and hashing

- Dictionaries are implemented using **hashtables**
- A **hash** is a function that takes a value and returns an integer.
- Dictionaries use hashes of the keys as indexes.
- This only works if keys are immutable.
- If a key were mutable, the hashtable would have to change every time the key changed.
- Lists are mutable, so cannot be keys.
- Dictionaries are mutable, so cannot be keys.
- Both lists and dictionaries can be values.

Redundant work in naive implementations

```
1 def fibonacci(n):
2     if n < 2:
3         return n
4     else:
5         return fibonacci(n-1) + fibonacci(n-2)
```



Memoizing

```
known = {0:0, 1:1}

def fibonacci(n):
    if n in known:
        return known[n]

    res = fibonacci(n-1) + fibonacci(n-2)
    known[n] = res
    return res
```

Memoizing

```
known = {0:0, 1:1}

def fibonacci(n):
    if n in known:
        return known[n]

    res = fibonacci(n-1) + fibonacci(n-2)
    known[n] = res
    return res
```

Uses global variable.

We'll see later how to get rid of that.

Global variables

```
1  x = 10
2  def show_global():
3      print(x)
4  show_global()
5  print(x)
6  def mod_global_bad():
7      x = 20
8      print(x)
9  mod_global_bad()
10 print(x)
11 def mod_global_good():
12     global x
13     x = 20
14     print(x)
15 mod_global_good()
16 print(x)
```

```
1  10
2  10
3  20
4  10
5  20
6  20
```

Good use of global variables

```
1 verbose = True
2
3 def example1():
4     if verbose:
5         print('Running example1')
```

What's wrong with this example?

```
1 been_called = False
2
3 def example2():
4     been_called = True          # WRONG
```

What's wrong with this example?

```
1 count = 0
2
3 def example3():
4     count = count + 1            # WRONG
```

Global mutable objects can be mutated locally

```
known = {0:0, 1:1}

def example4():
    known[2] = 1
```

Globals

- Can be confusing
- Declare them in functions with `global`
- Avoid using them

Debugging

- Scale down the input.
- Check summaries of outputs.
- Check types of outputs
- Write sanity checks:
  - averages should not be larger than maximum
  - averages should not be smaller than the minimum
- Write consistency checks
  - Number of sentences $<$ number of words
- Format the output so it's easy to spot errors
  - check out `pprint`

Vocabulary

mapping: A relationship in which each element of one set corresponds to an element of another set.

dictionary: A mapping from keys to their corresponding values.

key-value pair: The representation of the mapping from a key to a value.

item: In a dictionary, another name for a key-value pair.

key: An object that appears in a dictionary as the first part of a key-value pair.

value: An object that appears in a dictionary as the second part of a key-value pair. This is more specific than our previous use of the word "value".

Vocabulary

implementation: A way of performing a computation.

hashtable: The algorithm used to implement Python dictionaries.

hash function: A function used by a hashtable to compute the location for a key.

hashable: A type that has a hash function. Immutable types like integers, floats and strings are hashable; mutable types like lists and dictionaries are not.

lookup: A dictionary operation that takes a key and finds the corresponding value.

reverse lookup: A dictionary operation that takes a value and finds one or more keys that map to it.

Vocabulary

raise statement: A statement that (deliberately) raises an exception.

singleton: A list (or other sequence) with a single element.

call graph: A diagram that shows every frame created during the execution of a program, with an arrow from each caller to each callee.

memo: A computed value stored to avoid unnecessary future computation.

Vocabulary

global variable: A variable defined outside a function. Global variables can be accessed from any function.

global statement: A statement that declares a variable name global.

flag: A boolean variable used to indicate whether a condition is true.

declaration: A statement like global that tells the interpreter something about a variable.