COMP 204

Dictionaries

Yue Li
based on material from Mathieu Blanchette, Carlos Oliver Gonzalez and Christopher Cameron

Quiz 13 password

A very useful type: Dictionary

- A dictionary is said to be a *mapping* type because it maps *key* objects to *value* objects.
- Dictionaries are immensely useful and are the magic behind a lot of Python functionality
- Syntax:

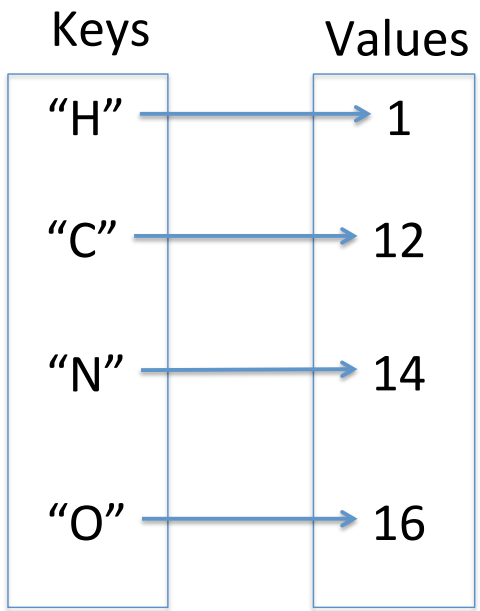
```
1 my_dict = { [key1]: [value1], [key2]: [value2],...}
```

- The analogy to a real dictionary works. The word you look up is the **key** and the definition is the **value**

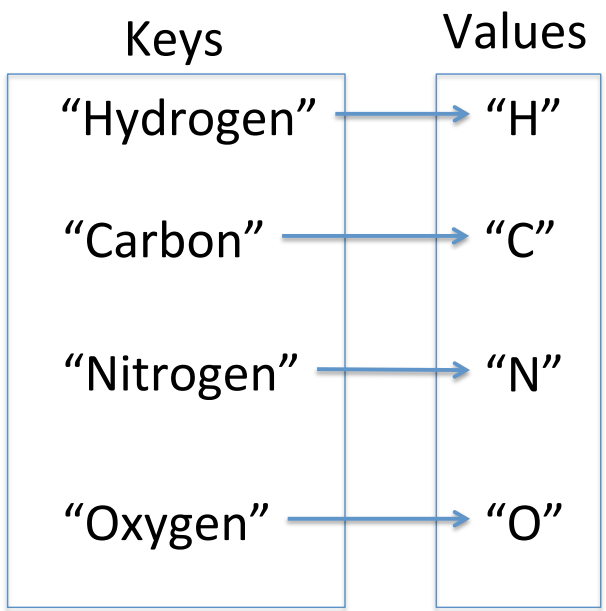
```
# this dictionary maps strings to integers
periodicTable = {"H":1, "C":12, "N":14, "O": 16}

elementsCodes = {"Hydrogen":"H", "Carbon":"C",
                 "Nitrogen":"N", "Oxygen":"O"}
```

periodicTable dictionary:



elementCodes dictionary:



Accessing elements in a dictionary

- Syntax: myDict[key] = value
- If key does not already exist in the dictionary, the new key/value pair is added
- If the key already exists, its previous value is overwritten

Deleting key/values: `del myDict[key]`

```
1  # this dictionary maps strings to integers
2  periodicTable = {"H":1, "C":12, "N":14, "O": 16}
3
4  elementsCodes = {"Hydrogen":"H", "Carbon":"C",
5                   "Nitrogen":"N", "Oxygen":"O"}
6
7  mass = periodicTable["C"]
8  mass = periodicTable["K"] # error key does not exist
9
10 periodicTable["He"] = 4  # adds key "He" with value 4
11 periodicTable["Na"]= 23  # adds key "Na" with value 23
12
13 #periodicTable now contains 6 keys,value pairs
14
15 periodicTable["C"] = 12.01  # overwrites value for key "C"
16
17 del periodicTable["N"]  # deletes key "N" and its value 14
```

About keys and values

Keys:

- Have to be immutable objects: int, float, str, tuple.
- Have to be unique in the dictionary: A dictionary cannot contain two elements with the same key.

Values:

- Values can be any type of object: int, float, str, tuple, list, dictionary, etc.
- Many keys can map to the same value

A dictionary can contain keys of many different types, and values of many different types:

```
# a dictionary with keys and values of different types
mixedDict = {"H":"Hydrogen", 17:"prime",
             30:[1,2, 3, 5], (4,5):20}


product = mixedDict[(4,5)]  # 20
primeFactors = mixedDict[30]     # [1,2, 3, 5]

fac = mixedDict[20]   # KeyError: 20 not in mixedDict
```

Dictionaries of dictionaries

The values stored in a dictionary can themselves be dictionaries!

```
# a dictionary where each value is itself a dictionary
periodicTable = {"H": {"name":"Hydrogen", "mass":1},
                 "C": {"name":"Carbon", "mass":12},
                 "N": {"name":"Nitrogen", "mass":14},
                 "O": {"name":"Oxygen", "mass":16} }

carbonDic = periodicTable["C"]  # {"name":"Carbon", "mass":12}
m = carbonDic["mass"]  # 12

#or more directly
m = periodicTable["C"]["mass"]  #12
```

Iterating through dictionaries

The function keys() returns the keys present in the dictionary.

```
per = {"H":1, "C":12, "N":14, "O": 16}

keyList = list( per.keys() ) # ["H", "C", "N", "O"]
# Note: the keys() function returns an object of
#       type dict_keys. This object is converted to a
#       list using the list() function

for k in keyList:
    print("Key",k,"has value",per[k])
```

The function items() returns key/value tuples in the dictionary

```
per = {"H":1, "C":12, "N":14, "O": 16}

itemList = list( per.items() )
# Note: the items() function returns an object of
#       type dict_items. This object is converted to a
#       list using the list() function

# itemList is now a list of tuples:
# [('H', 1), ('C', 12), ('N', 14), ('O', 16)]

for k,v in itemList:
    print("Key",k,"has value",v)
```

More functions on dictionaries

To test if a key is present in a dictionary, use the in operator: key in myDict , which evaluates to True if key is in myDict.

```
periodic = {"H":1, "C":12, "N":14, "O": 16}

newElement = "Na"
if newElement in periodic:
    print("Na is already in the dictionary")
else:
    print("Na is not in the dictionary")
```

To add the content of one dictionary, use the update() function.

```
per = {"H":1, "C":12, "N":14, "O": 16}
newTable = {"Na":23, "K":39}

# Add the content of newTable to per
per.update(newTable) # per now has 6 elements
                     # newTable still has 2
print(per) # {'H': 1, 'C': 12, 'N': 14, 'O': 16, 'Na': 23, '
    K': 39}
```

For more functions on dictionaries:

https://docs.python.org/3/library/stdtypes.html#mapping-types-dict

Example 1 counting unique nucleic acids in DNA sequence

Python file: counting.py

```
1  sequence = "ACGACTACGT"
2  nucleotides = "ACGT"
3  counts = {}  # an empty dictionary
4  for nuc in sequence:
5      if nuc in nucleotides:
6          if (nuc in counts)==False:
7              counts[nuc] = 1
8          else:
9              counts[nuc] += 1
10
11 print(counts)
```

Example 2 Compute the mass of molecules

Goal: Compute the mass of a molecule based on its chemical compostion. Assume that you have access to a dictionary of atomic masses.

Python file: mass.py

```
1 periodicTable = {"H":1, "C":12, "N":14, "O": 16}
2
3 aceticAcid = "CHHHCOOH"
4
5 mass = 0
6 for element in aceticAcid:
7     mass += periodicTable[element]
8
9 print("Mass of acetic acid is",mass)
```

Example 3 Create a dictionary with molecule name & mass

Goal: Create a dictionary using as keys the english name of molecules and as values their molar mass. Assume that you have access to a dictionary of atomic masses and a dictionary of chemical compositions:
Python file: mass2.py

```
1  periodicTable = {"H":1, "C":12, "N":14, "O": 16}
2
3  molecules = {"Carbon dioxyde":"COO",
4               "Nitric oxyde":"NO",
5               "Acetic acid":"CHHHCOOH"}
6
7  moleculeMass = {}  # the new dictionary we are about
8                     # to populate with name/mass pairs
9  for name, composition in molecules.items():
10     mass = 0
11     for atom in composition:
12         mass += periodicTable[atom]
13     moleculeMass[name] = mass
14
15 print(moleculeMass)
```

Some pre-release note about Assignment 2

- Released at midnight today (Feb 5 12 AM)
- Needleman-Wunsch global sequence alignment

Saul B Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins.
Journal of molecular biology, 48(3):443–453, 1970.

Introduction to sequence alignment

Suppose we want align the following two DNA sequences:

```
seq1:  GATTACA
seq2:  GCATGCT
```

For example, one possible alignment is:

```
G-ATTACA
GCA-TGCT
```

where "-" means a gap for an insertion or deletion (indel).
We will need a scoring system to evaluate the alignment quality:

- Match: $+2$
- Mismatch: -2
- Indel: -1

```
G-ATTACA
GCA-TGCT
```

$$2 - 1 + 2 - 1 + 2 - 2 + 2 - 2 = 2$$

Sequence alignment scoring grid

To compute the optimal alignment, we first construct a grid as shown below:

	-	G	C	A	T	G	C	T
-								
G								
A								
T								
T								
A								
C								
A								

Alignment scoring grid

Start with 0 in index (0,0) in the grid, because the first letter is a gap, we start by filling out the first row and first column by moving right and moving down, respectively. Adding the gap score of -1 for each shift, the results in the first row and the first column is shown below:

	-	G	C	A	T	G	C	T
-	0	-1	-2	-3	-4	-5	-6	-7
G	-1							
A	-2							
T	-3							
T	-4							
A	-5							
C	-6							
A	-7							

Divide-and-conquer: dividing alignment into smaller tasks

To fill out cell in index (1,1), the neighboring cells are shown below:

	-	G
-	0	-1
G	-1	?

We have three possible candidate sums:

- move diagonal: the diagonal top-left neighbor has score 0. The pairing of G and G is a match, so add the score for match: $0+2 = 2$
- move right: the left neighbor has score -1, represents an indel and produces (-1) + (-1) = -2.
- move down: the top neighbor has score -1 and moving from there represents an indel, so add the score for indel: (-1) + (-1) = (-2)

The best move is to **move diagonal** resulting the highest score 2.

The best move is to **move diagonal** resulting the highest score 2.

	-	G	C	A	T	G	C	T
-	0	-1	-2	-3	-4	-5	-6	-7
G	-1	2						
A	-2							
T	-3							
T	-4							
A	-5							
C	-6							
A	-7							

To fill out cell in index(1,2), the neighboring cells are shown below:

	-	G	C
-	0	-1	-2
G	-1	2	?

We have three possible candidate sums:

- move diagonal: the diagonal top-left neighbor has score 0. The pairing of G and C is a mismatch, so add the score for the mismatch: (-1) + (-2) = -3
- move right: the left neighbor has score 1, represents an indel and produces 2 + (-1) = 1.
- move down: the top neighbor has score -2 and moving from there represents an indel, so add the score for indel: (-2) + (-1) = (-3)

The best move is to **move right**

Let's do another one. To fill out cell in index(2,1), the neighboring cells are shown below:

	-	G
-	0	-1
G	-1	2
A	-2	?

Let's do another one. To fill out cell in index(2,1), the neighboring cells are shown below:

	-	G
-	0	-1
G	-1	2
A	-2	?

We have three possible candidate sums:

- move diagonal: the diagonal top-left neighbor has score -1. The pairing of A and G is a mismatch, so add the score for the mismatch: (-1) + (-2) = -3
- move right: the left neighbor has score -2, and moving from there represents an indel and produces -2 + (-1) = -3.
- move down: the top neighbor has score -2 and moving from there represents an indel, so add the score for indel: (2) + (-1) = 1

The best move is to **move down**

General sequence alignment algorithm

At any given index [i,j], we do the following to fill out the cell in

- move_diagonal = alignmentScoreGrid[i-1,j-1] + match_score
- move_down = alignmentScoreGrid[i-1, j] + gap_score
- move_right = alignmentScoreGrid[i, j-1] + gap_score
- alignmentScoreGrid[i,j] = max(move_diagonal, move_down, move_right)

Seq2

Seq1	-	G	C	A	T	G	C	T
-	0	-1	-2	-3	-4	-5	-6	-7
G	-1	2	1	0	-1	-2	-3	-4
A	-2	1	0	3	2	1	0	-1
T	-3	0	-1	2	5	4	3	2
T	-4	-1	-2	1	4	3	2	5
A	-5	-2	-3	0	3	2	1	4
C	-6	-3	0	-1	2	1	4	3
A	-7	-4	-1	2	1	0	3	2

Trace back the optimal alignments

Seq2

Seq1	-	G	C	A	T	G	C	T
-	0	-1	-2	-3	-4	-5	-6	-7
G	-1	2	1	0	-1	-2	-3	-4
A	-2	1	0	3	2	1	0	-1
T	-3	0	-1	2	5	4	3	2
T	-4	-1	-2	1	4	3	2	5
A	-5	-2	-3	0	3	2	1	4
C	-6	-3	0	-1	2	1	4	3
A	-7	-4	-1	2	1	0	3	2

Solution 1

Seq1: G - A T T A C A

Seq2: G C A - T G C T

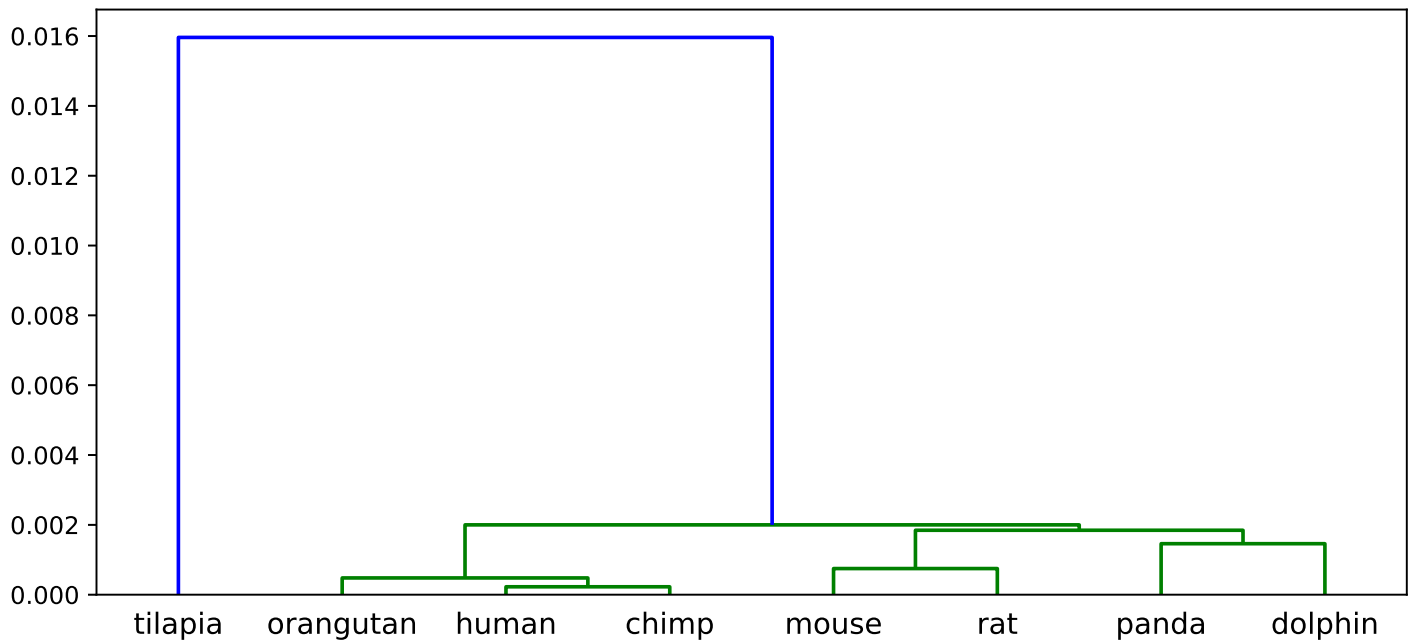
Solution 2

Seq1: G - A T T A C A

Seq2: G C A T - G C T

Sequence similarity of 8 homologs for *HIST1H1A*

	human	chimp	orangutan	mouse	rat	panda	dolphin	tilapia
human	430	418	398	260	267	298	287	114
chimp	418	430	394	268	275	306	295	114
orangutan	398	394	430	252	259	286	283	118
mouse	260	268	252	426	381	296	269	104
rat	267	275	259	381	428	299	268	111
panda	298	306	286	296	299	430	319	110
dolphin	287	295	283	269	268	319	436	119
tilapia	114	114	118	104	111	110	119	446



Conserved residues in the human HIST1H1A sequence

MSETVPPAPAASAAPEKPLAGKKAKKPAKAAAASKKKPAGPSVSELIVQA

6666264626352356662636345666626324666566666666666

ASSSKERGGVSLAALKKALAAAGYDVEKNNSRIKLGIKSLVSKGTLVQTK

2666666266666666466566666666666666256664666666666

GTGASGSFKLNKKASSVETKPGASKVATKTKATGASKKLKKATGASKKSV

6666466665666614352632266316262334666166362626626

KTPKKAKKPAATRKSSKNPKKPKTVKPKKVAKSPAKAKAVKPKAAKARVT

6665466663624566266651564665566666666666565066

KPKTAKPKKAAPKKK

666666666666666

Number indicates the number of species (6 species at max, excluding tilapia) that share common residue with the human sequence