DICTIONARIES

DICTIONARIES

- What is a dictionary?
  - You can think if a dictionary as a collection of pairs, where the first element of each pair, the ***key***, is used to retrieve the second element, the ***value***
    - These pairs are usually called key-value pairs
  - Thus we ***map a key to a value***

DICTIONARIES

- The key acts as an index to find the associated value
- Just like a real world dictionary, you look up a word by its spelling to find the associated definition
- A dictionary can be searched to locate the value associated with a key

DICTIONARIES

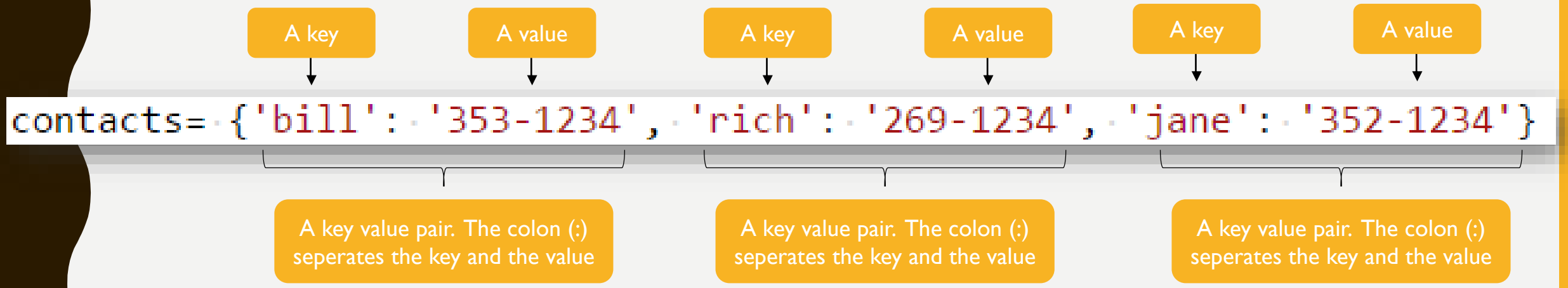
- In Python we often refer to dictionaries as a dict
  - We can create dictionaries using the curly braces or the dict constructor

Here are two ways of creating an empty dictionary

```
1    empy_dict = {}
2    another_empty_dict = dict()
```

DICTIONARIES

- We create a dictionary with values like this
  - Each key-value pair is separated by a comma just like values in a list or a tuple



DICTIONARIES

- Here are two ways of creating the same dictionary
  - The only difference is the format
  - If the dictionary contains many initial key-value pairs it may be more readable to format the dictionary like is done in the latter example

```
contacts= {'bill': '353-1234', 'rich': '269-1234', 'jane': '352-1234'}
```

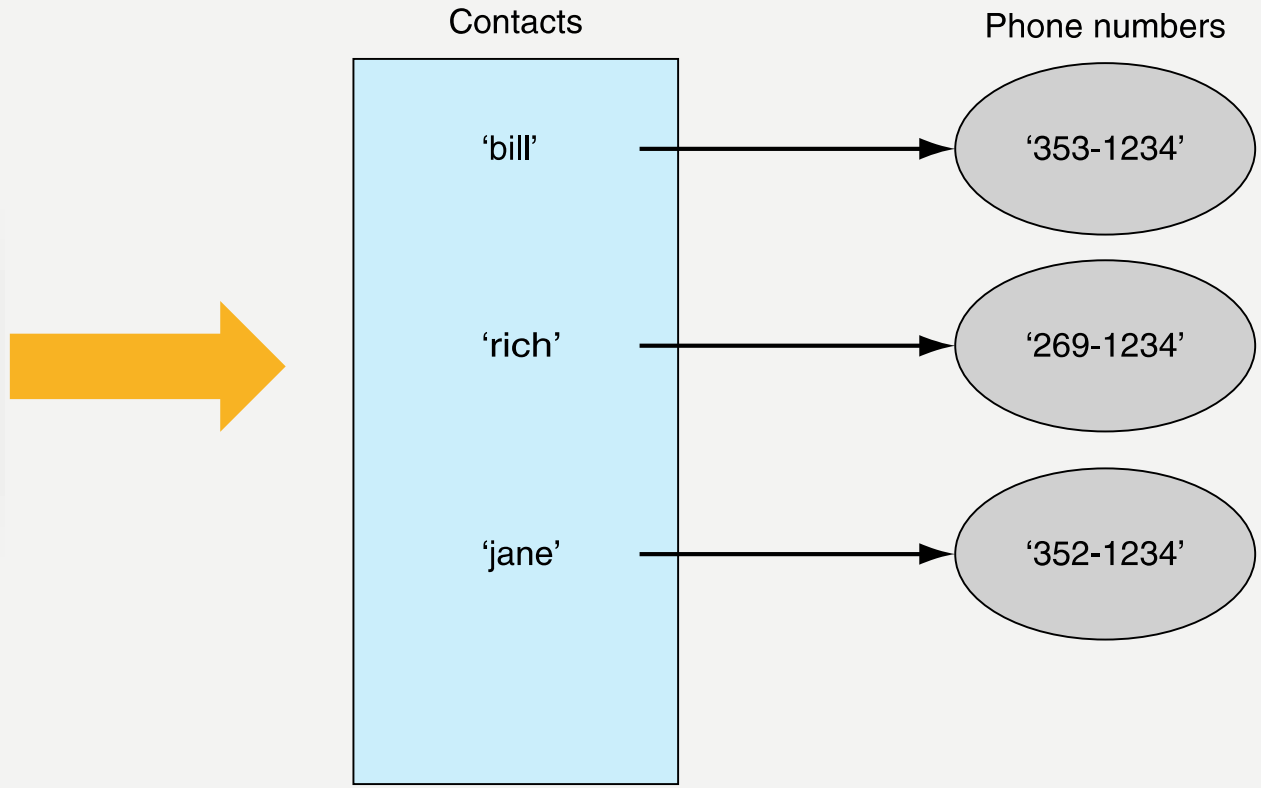
These two code snippets create the same dictionary. The only difference is the format

```
1  contacts= {
2      'bill': '353-1234',
3      'rich': '269-1234',
4      'jane': '352-1234'
5  }
```

DICTIONARIES

- Here is a visual representation of the dictionary created in the code snippet

```
1   contacts= {
2       'bill': '353-1234',
3       'rich': '269-1234',
4       'jane': '352-1234'
5   }
```



DICTIONARIES

- A key must be immutable!
- strings, integers, tuples are fine as keys
  - lists are NOT
- The value can be any type

In this code snippet all the keys are string and all the values are strings as well. We see that because of the single quotes

```
contacts= {
    'bill': '353-1234',
    'rich': '269-1234',
    'jane': '352-1234'
}
```

DICTIONARIES

- Dictionaries are collections but do note that they are not sequences like lists, strings or tuples
  - there is no order to the elements of a dictionary
    - in fact, the order (for example, when printed) might change as elements are added or deleted

DICTIONARIES

- How do we access values in a dictionary?
  - We use the [ ] operator just like we do with lists, tuples and strings
  - The difference is that we use a key as the index!

Here we are accessing the value that is associated with the key bill and printing it to the screen

Here we are accessing the value that is associated with the key jane and printing it to the screen

```
1  contacts= {
2      'bill': '353-1234',
3      'rich': '269-1234',
4      'jane': '352-1234'
5  }
6
7  print(contacts['bill']) # prints '353-1234'
8  print(contacts['jane']) # prints '352-1234'
```

Do note that if the key does not exist in the dictionary an error will be raised!

DICTIONARIES

- How do we add a key-value pair to a dictionary?
  - We can use the square brackets!

We can use the square brackets like this. This will add a key of robert to the contacts dictionary with the value of 581-2345

```
1  contacts= {
2      'bill': '353-1234',
3      'rich': '269-1234',
4      'jane': '352-1234'
5  }
6
7  contacts['robert'] = '581-2345'
```

DICTIONARIES

- Like lists, dictionaries are a mutable data structure
  - you can change the object via various operations, such as index assignment

This means that if there is already a key of robert within the dictionary it will simply be updated. A new key value pair with only be added if the key is not already in the dictionary!

```
1  contacts= {
2      'bill': '353-1234',
3      'rich': '269-1234',
4      'jane': '352-1234'
5  }
6
7  contacts['robert'] = '581-2345'
```

DICTIONARIES

- Here is another example of how dictionaries are immutable

```
1   my_dict = {
2       'bill':3,
3       'rich':10
4   }
5   print(my_dict['bill']) # prints 2
6
7   my_dict['bill'] = 100
8
9   print(my_dict['bill']) # prints 100
```

DICTIONARIES

Like other data structures we've seen so far, dictionaries respond to these functions

- `len(my_dict)`
  - number of key : value **pairs** in the dictionary
- `element in my_dict`
  - boolean, is `element` a **key** in the dictionary
- `for key in my_dict:`
  - iterates over the **keys** of a dictionary

```
1   my_dict = {
2       'bill':3,
3       'rich':10
4   }
5
6   length = len(my_dict) # returns 2
7
8   if 'bill' in my_dict:
9       print("value is in the dictionary")
10
11  for key in my_dict:
12      print(key)
```

DICTIONARIES

- Here are some methods dictionaries have to access the keys and values
  - `my_dict.items()` – all the key/value pairs
  - `my_dict.keys()` – all the keys
  - `my_dict.values()` – all the values
    - They return what is called a *dictionary view*

```
my_dict = {
    'bill':3,
    'rich':10,
    'jane': 23
}


print(my_dict.items()) # dict_items([('bill', 3), ('rich', 10), ('jane', 23)])
print(my_dict.values()) # dict_values([3, 10, 23])
print(my_dict.keys()) # dict_keys(['bill', 'rich', 'jane'])
```

DICTIONARIES

- We can use all these methods with the for loop
  - `my_dict.items()`
  - `my_dict.keys()`
  - `my_dict.values()`

```
1  my_dict = {
2      'bill':3,
3      'rich':10,
4      'jane': 23
5  }
6
7  for value in my_dict.values():
8      print(value)
9
10 for key in my_dict.keys():
11     print(key)
12
13 for key_value in my_dict.items():
14     print(key_value)
15
16 for key, value in my_dict.items():
17     print(key, value)
```

Here the variable key_value will hold a tuple for each key-value pair

Note that here we use tuple unpacking for each item in the dictionary

DICTIONARIES

Dictionaries have several other methods we can use

- `my_dict.clear()` – empties the dictionary
- `my_dict.copy()` - shallow copy
- `my_dict.pop(key)` – removes the key and value associated with the given key, it also returns the value associated with the given key

DICTIONARIES
BUILDING DICTIONARIES FASTER

- `zip` creates pairs from two parallel lists
  - It will always use the shorter list as a limit

```
1   list_of_tuples = zip("abc",[1,2,3]) #[('a',1),('b',2),('c',3)]
2
3   dict(zip("abc",[1,2,3])) #{'a': 1, 'c': 3, 'b': 2}
```

DICTIONARIES

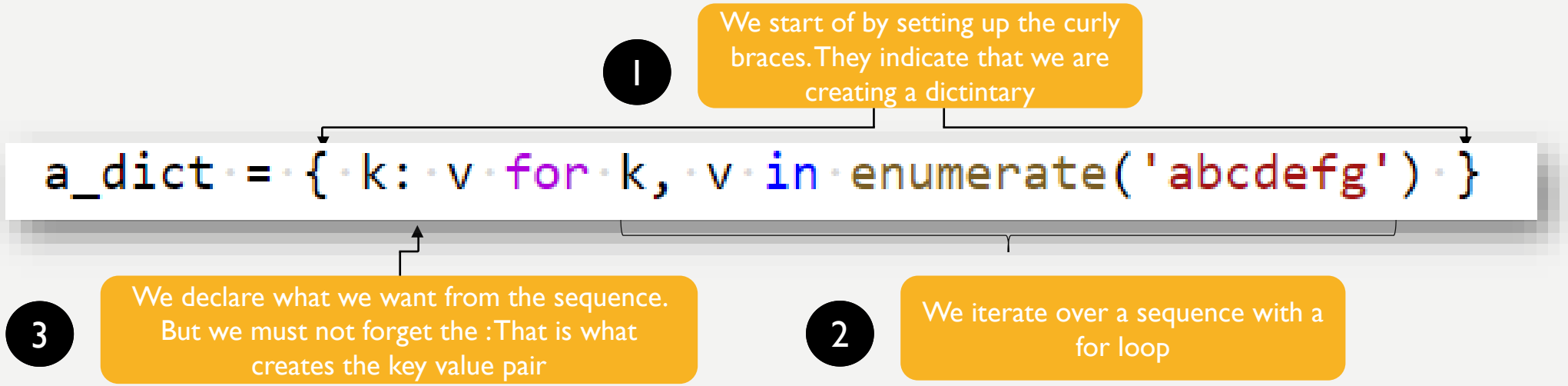
- Like list comprehensions, you can write shortcuts that generate a dictionary with the same control you had with list comprehensions
  - Dictionaries are enclosed with `{}` (remember, list comprehensions use the `[]`)
  - Dictionaries always have a key-value pair with a : between the key and the value

DICTIONARIES
DICT COMPREHENSION

Enumerate is similar to the range function. The difference is that enumerate gives you a value and the index of the value in the enueration(sequence)

```
1   a_dict = { k: v for k, v in enumerate('abcdefg') }
2
3   print(a_dict) # {0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e', 5: 'f', 6: 'g'}
```

DICTIONARIES COMPREHENSION



FREQUENCY OF WORDS IN LIST 3 WAYS

DICTIONARIES MEMBERSHIP TEST

```
count_dict = {}
for word in word_list:
    if word in count_dict:
        count_dict[word] += 1
    else:
        count_dict[word] = 1
```

DICTIONARIES EXCEPTIONS

```
count_dict = {}
for word in word_list:
    try:
        count_dict[word] += 1
    except KeyError:
        count_dict[word] = 1
```

DICTIONARIES
THE GET METHOD

The `get` method returns the value associated with a dict key or a default value provided as second argument. Below, the default is 0

```
count_dict = {}
for word in word_list:
    count_dict[word] = count_dict.get(word,0) + 1
```

DICTIONARIES 4 FUNCTIONS

- `add_word(word, word_dict)`. Add word to the dictionary. No return
- `process_line(line, word_dict)`. Process line and identify words. Calls `add_word`. No return.
- `pretty_print(word_dict)`. Nice printing of the dictionary contents. No return
- `main()`. Function to start the program.

DICTIONARIES PASSING MUTABLES

- Because we are passing a mutable data structure, a dictionary, we do not have to return the dictionary when the function ends
- If all we do is update the dictionary (change the object) then the argument will be associated with the changed object.

DICTIONARIES

```
1 def add_word(word, word_count_dict):
2     '''Update the word frequency: word is the key, frequency is the value.'''
3     if word in word_count_dict:
4         word_count_dict[word] += 1
5     else:
6         word_count_dict[word] = 1
```

DICTIONARIES

```
import string
def process_line(line, word_count_dict):
    '''Process the line to get lowercase words to add to the dictionary.'''
    line = line.strip()
    word_list = line.split()
    for word in word_list:
        # ignore the '--' that is in the file
        if word != '--':
            word = word.lower()
            word = word.strip()
            # get commas, periods and other punctuation out as well
            word = word.strip(string.punctuation)
            add_word(word, word_count_dict)
```

DICTIONARIES

```
1  def pretty_print(word_count_dict):
2      '''Print nicely from highest to lowest frequency.'''
3      # create a list of tuples, (value, key)
4      # value_key_list = [(val,key) for key,val in d.items()]
5      value_key_list=[]
6      for key,val in word_count_dict.items():
7          value_key_list.append((val,key))
8      # sort method sorts on list's first element, the frequency.
9      # Reverse to get biggest first
10     value_key_list.sort(reverse=True)
11     # value_key_list = sorted([(v,k) for k,v in value_key_list.items()],
   reverse=True)
12     print('{:11s}{:11s}'.format('Word', 'Count'))
13     print('_'*21)
14     for val,key in value_key_list:
15         print('{:12s}  {:<3d}'.format(key,val))
```

DICTIONARIES

```
1 def main ():
2     word_count_dict={}
3     gba_file = open('gettysburg.txt','r')
4     for line in gba_file:
5         process_line(line, word_count_dict)
6     print('Length of the dictionary:',len(word_count_dict))
7     pretty_print(word_count_dict)
```

PERIODIC TABLE EXAMPLE

COMMA SEPARATED VALUES (CSV)

- ***csv*** files are a text format that are used by many applications (especially spreadsheets) to exchange data as text
- row oriented representation where each line is a row, and elements of the row (columns) are separated by a comma
- despite the simplicity, there are variations and we'd like Python to help

DICTIONARIES AND THE CSV MODULE

- `csv.reader()` takes an opened file object as an argument and reads one line at a time from that file
- Each line is formatted as a list with the elements (the columns, the comma separated elements) found in the file

DICTIONARIES
ENCODINGS OTHER THAN UTF-8

- this example uses a csv file encoded with characters other than UTF-8 (our default)
  - in particular, the symbol ± occurs
- can solve by opening the file with the correct encoding, in this case `windows-1252`

DICTIONARIES

```
>>> import csv
>>> periodic_file = open("Periodic-Table.csv", "r",encoding="windows-1252")
>>> reader = csv.reader(periodic_file)
>>> for row in reader:
        print(row)

# some of the output data
['2', 'He', '18', 'VIII A', '1', 'helium', '4.003', '0', '', '', ...]
['3', 'Li', '1', 'I A', '2', 'lithium', '6.941', '+1', '', '', ... ]
['4', 'Be', '2', 'II A', '2', 'beryllium', '9.012', '+2', '', '', ...]
['5', 'B', '13', 'III A', '2', 'boron', '10.81', '+3', '', '', ...]
# etc. etc.
```

```
import csv

def read_table(a_file, a_dict):
    """Read Periodic Table file into a dict. with element symbol as key.
        periodic_file is a file object opened for reading """
    data_reader = csv.reader(a_file)

    for row in data_reader:
        # ignore header rows: elements begin with a number
        if row[0].isdigit():
            symbol_str = row[1]
            a_dict[symbol_str] = row[:8]  # ignore end of row

def parse_element(element_str):
    """Parse element string into symbol and quantity,
       e.g. Si2 returns ('Si',2)"""
    symbol_str=""
    quantity_str = ""
    for ch in element_str:
        if ch.isalpha():
            symbol_str = symbol_str + ch
        else:
            quantity_str = quantity_str + ch
    if quantity_str == "":                    # if no number, default is 1
        quantity_str = "1"
    return symbol_str, int(quantity_str)
```

```
# 1. Read File
periodic_file = open("Periodic-Table.csv", "r",encoding="windows-1252")

# 2. Create Dictionary of Periodic Table using element symbols as keys
periodic_dict={}
read_table(periodic_file, periodic_dict)

# 3. Prompt for input and convert compound into a list of elements
compound_str = input("Input a chemical compound, hyphenated, e.g. C-O2: ")
compound_list = compound_str.split("-")

# 4. Initialize atomic mass
mass_float = 0.0
print("The compound is composed of: ", end=' ')

# 5. Parse compound list into symbol-quantity pairs, print name, and add mass
for c in compound_list:
    symbol_str, quantity_int = parse_element(c)
    print(periodic_dict[symbol_str][5], end=' ')  # print element name
    mass_float = mass_float + quantity_int *\
                 float(periodic_dict[symbol_str][6]) # add atomic mass

print("\n\nThe atomic mass of the compound is", mass_float)

periodic_file.close()
```