Dictionaries

A dictionary is a collection of key-value pairs that allows access to the value by key (as opposed to access by index as in a list).

Let's look at some examples:

In [1]:
```
# first, let's create an empty dictionary
codons = {}
# then let's add some items to it
codons['AUG'] = 'Met'
codons['UAU'] = 'Tyr'
codons
```

Out[1]:
```
{'AUG': 'Met', 'UAU': 'Tyr'}
```

Note the key, value relationship:

```
dictionary_name[key] = value
{key: value, key: value}
```

To get the value associated with a key we can use square brackets:

In [2]:
```
codons['AUG']
```

Out[2]:
```
'Met'
```

You can create a dictionary by providing key-value pairs in the same format as the output from above:

In [10]:
```
codons = {'GCU': 'Ala', 'AUG': 'Met', 'UAU': 'Tyr', 'CCA': 'Pro'}
codons
```

Out[10]:
```
{'GCU': 'Ala', 'AUG': 'Met', 'UAU': 'Tyr', 'CCA': 'Pro'}
```

In [15]:
```
numbers = {1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five'}
numbers
```

Out[15]:
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four', 5: 'five'}
```

Note that order doesn't matter and is not fixed (starting in Python 3.6 the order is fixed)!

The `len` function returns the number of key-value pairs:

In [16]:
```
len(numbers)
```

Out[16]:
```
5
```

We can change the values in a dictionary by referencing its key:

In [19]:
```
codons = {'GCU': 'Ala', 'AUG': 'Met', 'UAU': 'Tyr', 'CCA': 'Pro'}
codons['AUG'] = 'Start'
codons
```

Out[19]:
```
{'GCU': 'Ala', 'AUG': 'Start', 'UAU': 'Tyr', 'CCA': 'Pro'}
```

Recall that we can do something similar to lists by referencing an index:

In [21]:
```
t = [0, 1, 2]
t[1] = 'x'
t
```

Out[21]:
```
[0, 'x', 2]
```

If we want to know if a dictionary contains a particular key we can use the `in` operator:

In [22]:
```
codons = {'GCU': 'Ala', 'AUG': 'Met', 'UAU': 'Tyr', 'CCA': 'Pro'}
'AUG' in codons
```

Out[22]:
```
True
```

But using in to test if a dictionary contains a particular value doesn't work:

In [23]:
```
'Met' in codons
```

Out[23]:
```
False
```

Just like lists and strings, dictionaries have useful methods. For example, we can use the `values()` method to determine if a dictionary has a particular value:

In [27]:
```
'Met' in codons.values()
```

Out[27]:
```
True
```

Strings can be used as keys:

In [30]:
```
numbers  = {'one': 1, 'two': 2, 'three': 3}
numbers
```

Out[30]:
```
{'one': 1, 'two': 2, 'three': 3}
```

Integers can be also be used as keys:

In [31]:
```
numbers = {1: 'one', 2: 'two', 3: 'three'}
numbers
```

Out[31]:
```
{1: 'one', 2: 'two', 3: 'three'}
```

Lists can't be used a keys:

In [32]:
```
numbers = {[1,2]: 'onetwo'}
numbers
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-32-6e0ae8c1bdd1> in <module>()
----> 1 numbers = {[1,2]: 'onetwo'}
      2 numbers

TypeError: unhashable type: 'list'
```

But lists can be used as values:

In [34]:
```
numbers = {'onetwo': [1,2]}
numbers
```

Out[34]:
```
{'onetwo': [1, 2]}
```

You can iterate over a dictionary using a `for` loop:

In [35]:
```
comp_nt = {'A': 'T', 'C': 'G', 'G': 'C', 'T': 'A'}
for key in comp_nt:
    print(key + '-' + comp_nt[key])
```

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

Again, notice that the key-value pairs are not ordered prior to Python 3.6.

Dictionaries are extremely useful for efficiently keeping track of things. Let's write a function that computes the number of occurrences of all k-mers of length k in a sequence:

In [41]:
```
def kmer_count(sequence, k):
    counts = {}
    for i in range(len(sequence)- k + 1):
        kmer = sequence[i:i+k]
        if not(kmer in counts):
            counts[kmer] = 0
        counts[kmer] += 1
    return counts

kmer_count('ATGGGAGGCGGACATG', 3)
```

Out[41]:
```
{'ATG': 2,
 'TGG': 1,
 'GGG': 1,
 'GGA': 2,
 'GAG': 1,
 'AGG': 1,
 'GGC': 1,
 'GCG': 1,
 'CGG': 1,
 'GAC': 1,
 'ACA': 1,
 'CAT': 1}
```

The `get()` method returns a value for a given key:

```
dict.get(key, default_value = None)
```

In [37]:
```
codons = {'GCU': 'Ala', 'AUG': 'Met', 'UAU': 'Tyr', 'CCA': 'Pro'}
print(codons.get('GCU'))
print(codons['GCU'])
```

```
Ala
Ala
```

But the `get()` methods is especially useful for assigning default values when creating new key-value pairs if they don't already exist:

In [40]:
```
codons['AUG'] = codons.get('AUG', 'Start')
codons['AAA'] = codons.get('AAA', 'Lys')
codons
```

Out[40]:
```
{'GCU': 'Ala', 'AUG': 'Met', 'UAU': 'Tyr', 'CCA': 'Pro', 'AAA': 'Lys'}
```

Let's modify the `kmer_count()` function to use the `get` method:

In [44]:
```
def kmer_count(sequence, k):
    counts = {}
    for i in range(len(sequence)-k+1):
        kmer = sequence[i:i+k]
        # if not(kmer in counts):
        #     counts[kmer] = 0
        # counts[kmer] += 1
        counts[kmer] = counts.get(kmer, 0) + 1
    return counts

kmer_count('ATGGGAGGCGGACATG', 3)
```

Out[44]:
```
{'ATG': 2,
 'TGG': 1,
 'GGG': 1,
 'GGA': 2,
 'GAG': 1,
 'AGG': 1,
 'GGC': 1,
 'GCG': 1,
 'CGG': 1,
 'GAC': 1,
 'ACA': 1,
 'CAT': 1}
```

The second argument of the `get` method of a dictionary is the value which should be used in case the given key is not in the dictionary, which allowed us to skip the step that handles the case of a kmer that is not in the dictionary.

Exercises:

Answer key (http://rna.colostate.edu/dokuwiki/doku.php?id=script:ex9)

9a) Write a for loop to iterate over each key in the dictionary below and print the key, value pair in fasta format.

In [ ]:
```
fasta = {'let-7': 'TGAGGTAGTAGGTTGTATAGTT',
         'lin-4': 'TCCCTGAGACCTCAAGTGTGA',
         'miR-1': 'TGGAATGTAAAGAAGTATGTA'}
```

In [ ]:
```

```

9b) Write a function, `revcomp(sequence)` that uses dictionaries and slicing to compute the reverse complement of a sequence.

In [ ]:
```

```

9c) Write a function, `fasta_converter(fasta_file)` , that converts fasta formatted data containing short sequences each on a single line for example, c_elegans_mirnas.fa.gz (http://rna.colostate.edu/dokuwiki/doku.php?id=sample_data), to tab delimited format. The function should read in a fasta file (sample data here) with each sequence id and sequence as key-value pair in a dictionary and then write the key-value pairs to new file in tab delimited format (this is a precursor to the homework assignment):

```
INPUT
>sequence_id1
sequence1
>sequence_id2
sequence2

OUTPUT
sequence_id1    sequence1
sequence_id2    sequence2
```

In [ ]: