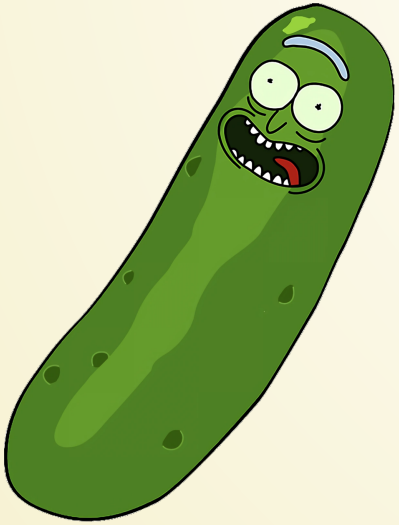
Dictionaries and Sets

and pickles!



adapted from material by Mike Scott and Bill Young at the University of Texas at Austin

Converting Numbers to Words

"13177284" is pronounced "one three one seven seven two eight four".

Write a program which accomplishes this conversion.

Small bite: convert a single int into its written equivalent.

Converting Words to Numbers

"13177284" is pronounced "one three one seven seven two eight four".

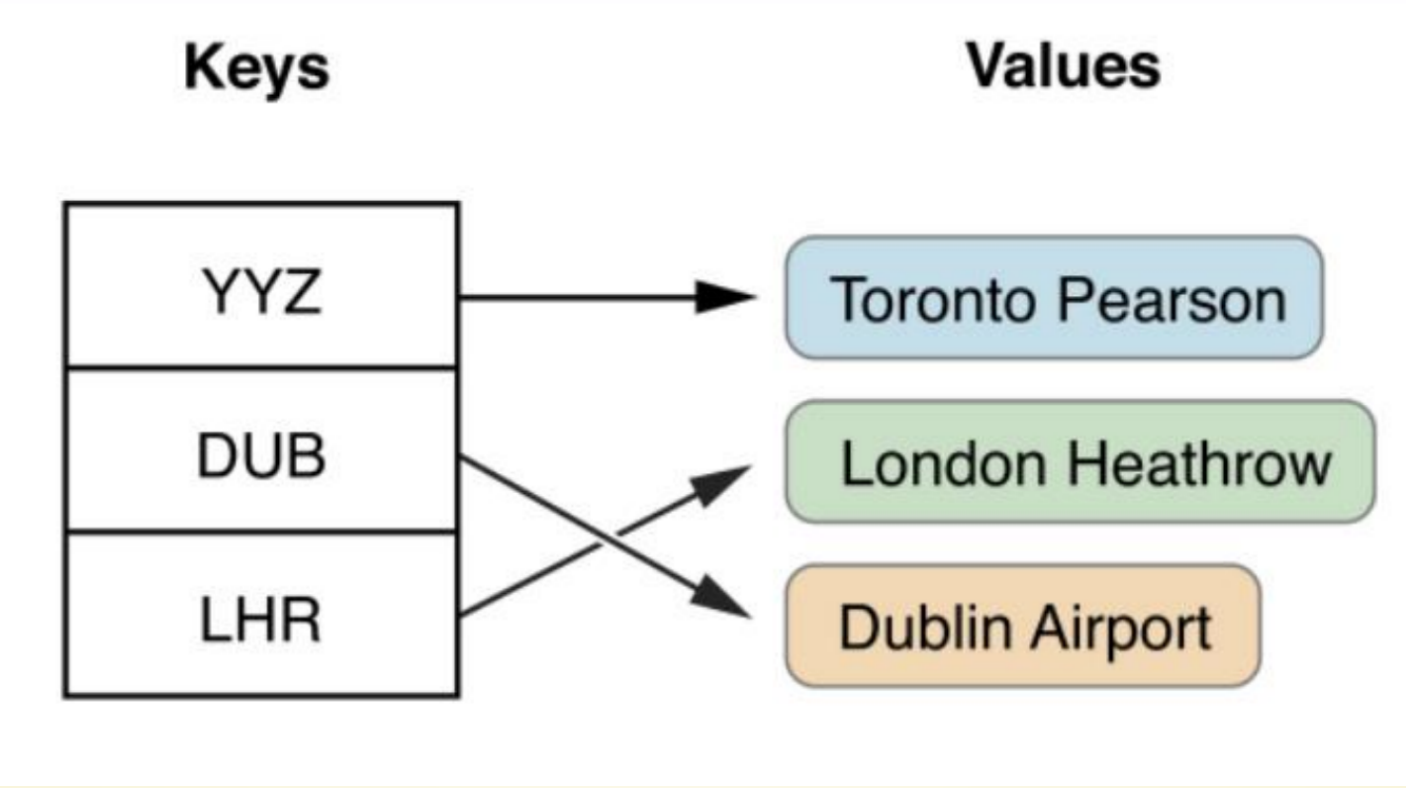
Write a program which, given the written-out form, converts to the numeric string (not the number!).

Dictionaries

Dictionaries give us *associative lookup*

A collection of *keys* and *values*. Each key has a unique value associated with it.

Given a key, you can quickly look up the value.



Constructing Dictionaries

Can use special syntax for it.

```
In [1]: dict1 = { "hey": 3, "help": 4, "power": 5 }

In [2]: dict1
Out[2]: {'hey': 3, 'help': 4, 'power': 5}

In [3]: dict1['hey']
Out[3]: 3
```

Or we can just add keys one at a time.

```
1  dict1 = dict()
2  dict1["hey"] = 3
3  dict1["help"] = 4
4  dict1["power"] = 5
```

Accessing Dictionaries

```
1 doubles = { 2:4, 3:6, 4:8, 5:10 }
2 print(doubles[2])
3 print(doubles[3])
4 print(doubles[6])
```

We can use [] notation to look up elements. But be careful! If the key isn't in the dictionary, we'll get a KeyError.

Can check if key is in dictionary ahead of time by using in or not in.

```
1 def safe_lookup(d, key):
2   if key in d:
3     return d[key]
4   else:
5     print(f"Wuh woh! {key} was not in the dictionary!")
6     return None
```

Adding Elements

```
dictionary[key] = value
```

- If key is not in the dictionary, adds the key to dictionary with the new value.
- If key is in the dictionary, modifies the current value with the new one.

Deleting Elements

```
del dictionary[key]
```

If key is not in the dictionary, raises a KeyError

Can also use dictionary.pop(key) to get the value.

Iterating

```
1 for key in dictionary.keys():
2   loop body
```

```
1 for value in dictionary.values():
2   loop body
```

```
1 for (key, value) in dictionary.items():
2   loop body
```

```
1 # Equivalent to first form
2 for key in dictionary:
3   loop body
```

Let's Write a Program

Write a program that counts how often each character occurs in a string.

Sanity check afterwards to make sure that the total number of counts is the same as the input length.

Sets

Sets in Python behave like mathematical sets:

- Unordered
- Elements can be different types
- **Every item is unique**

```
1  s = set()
2  s.add(2)
3  s.add(3)
4  s.add(5)
5  s.add(2)
```

Set Methods

Method	Description
add(x)	Add a single element to the set
update(c)	Add all elements from c into the set
remove(x)	Remove x from set, raising KeyError if not present
discard(x)	Remove x from the set, no error raised
union(s2)	Find the union of s1, s2
intersection(s2)	Find the intersection of s1, s2
issuperset(s2)	Is s1 a superset of s2?
issubset(s2)	Is s1 a subset of s2?

Super Neat Trick

```
1  def num_unique_elem(collection):
2      ???
```

```
1  def num_unique_elem(collection):
2      return len(set(collection))
```

Pickling

Sometimes, we want to save data to a file.

Could figure out how to write data to a representation, read it back.

For example 2D lists can often be written as a CSV.

Dictionaries can be written using JSON.

But what about frozenset? defaultdict?
custom data structures?

Python Pickles

Lets us save our data into files without worrying about the format! As long as we can write data to a file, we can get it back out later.

```
1 import pickle
2 # To save an object
3 with open("object.pkl", "wb") as pklfile:
4   pickle.dump(obj, pklfile)
5
6 # To get an object back
7 with open("object.pkl", "rb") as pklfile:
8   obj = pickle.load(pklfile)
```

Notes

Don't rely on pickle files long term! The internal format can change, meaning e.g. if you save a file with Python 3.8 and try to load with Python 3.11, you might get an error.

Pickles are easy, but sometimes different formats will be faster and better for specific data types (e.g. dicts are almost always better serialized in JSON, homogeneous lists of primitives are almost always better serialized in CSV).