Merriam-Webster (Dictionaries)

Hit **Space** to move forward and **Shift + Space** to move backward

What are dictionaries?

A **dictionary** is a data structure, or collection of values.

We've seen another type of data structure in the past before...

Lists!

```
person = ['George Merriam',
    'January 20, 1803',
    'Worcester, Massachusetts'
]

print(f'Name: {person[0]}')
print(f'Birthday: {person[1]}')
print(f'Birthplace: {person[2]}')
```

```
Name: George Merriam
Birthday: January 20, 1803
Birthplace: Worcester, Massachusetts
```

```
person = ['George Merriam',
    'January 20, 1803',
    'Worcester, Massachusetts'
]

print(f'Name: {person[0]}')
print(f'Birthday: {person[1]}')
print(f'Birthplace: {person[2]}')

person.insert(0, 'publisher') # What would happen?
```

```
Name: George Merriam
Birthday: January 20, 1803
Birthplace: Worcester, Massachusetts
```

When values are better organized by keys, rather than order (i.e. indices), use a dictionary.

And when are values better organized by keys? When each value has a specific *meaning* within a collection of values.

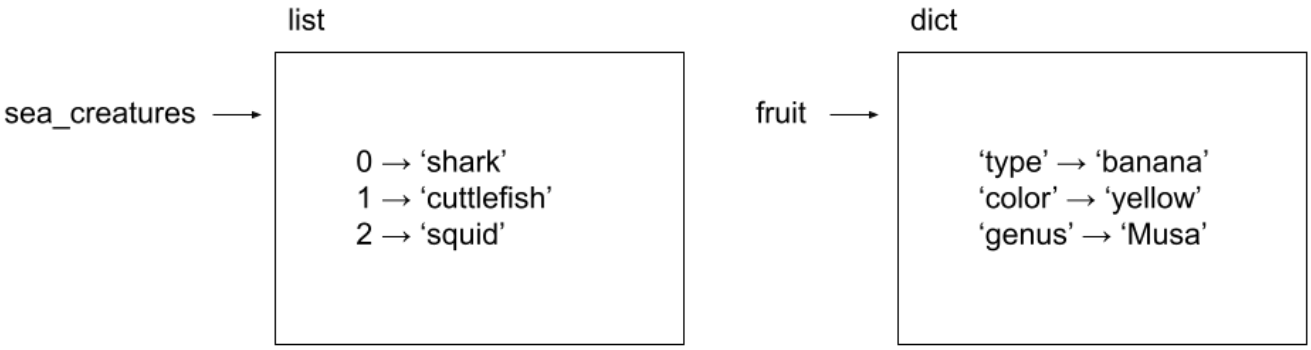
```
person = {
    'name': 'George Merriam',
    'birthday': 'January 20, 1803',
    'birthplace': 'Worcester, Massachusetts'
}

print(f'Name: {person["name"]}')
print(f'Birthday: {person["birthday"]}')
print(f'Birthplace: {person["birthplace"]}')
```

```
Name: George Merriam
Birthday: January 20, 1803
Birthplace: Worcester, Massachusetts
```

While values in a list are indexed by integers, values in a dictionary are indexed by keys.

```
sea_creatures = ['shark', 'cuttlefish', 'squid']
fruit = {
    'type': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
```



What can dictionaries be used for?

(So many things...)

Web applications

```
user = {
    'email': 'ykim@allegheny.edu',
    'first_name': 'Maria',
    'last_name': 'Heinert',
    'age': 25
}
```

Text prediction

```
after_i = {
    'am': 34,
    'like': 68,
    'use': 20,
    'think': 90
}
```

Creating dictionaries

Creating an empty dictionary

Use `dict()`.

```
fruit = dict()
print(type(fruit))
```

```
<class 'dict'>
```

Creating a dictionary with items

Enclose **items**, or key-value pairs, in curly braces `{}`. A key and its value should be separated by a colon `:`.

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
```

Are dictionaries ordered?

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print(fruit)
print(fruit)
print(fruit)
```

```
{'name': 'banana', 'color': 'yellow', 'genus': 'Musa'}
{'name': 'banana', 'color': 'yellow', 'genus': 'Musa'}
{'name': 'banana', 'color': 'yellow', 'genus': 'Musa'}
```

Python v.3.7: **2018**

Software is always changing!

Looking up a value by its key in a dictionary

Use square brackets `[]`.

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print(fruit['name'])
print(fruit['color'])
print(fruit['genus'])
```

```
banana
yellow
Musa
```

What if the key is not in the dictionary?

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print(fruit['price'])
```

```
---------------------------------------------------------------------------
-

KeyError                                  Traceback (most recent call
last)

/var/folders/bj/bw1mwdzj6vsbs4zf676n7rw80000gq/T/ipykernel_73978/378237777
2.py in <module>
      4     'genus': 'Musa'
      5 }
----> 6 print(fruit['price'])
```

```
KeyError: 'price'
```

Getting the number of items in a dictionary

Use the `len` function!

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print(len(fruit)) # Why not 6?
```

```
3
```

Checking if a key is in a dictionary

Use the `in` operator.

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print('name' in fruit)
print('price' in fruit)
```

```
True
False
```

Checking if a value is in a dictionary

Get the values using the `values` method and then use the `in` operator.

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
```

```
    'genus': 'Musa'
}
print('banana' in fruit.values())
print('genus' in fruit.values())
```

```
True
False
```

Quick review

A dictionary is a data structure whose values are indexed by keys, rather than by integers.

```
fruits = ['apple', 'banana', 'pineapple']
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
```

Create an empty dictionary with `dict()`.

```
empty = dict()
print(empty)
```

```
{}
```

Create a dictionary with items using curly braces `{}`.

Each key should be separated from its value by a colon `:`.

```
user = {
    'email': 'ykim@allegheny.edu',
    'first_name': 'Maria',
    'last_name': 'Heinert',
    'age': 25
}
```

Look up a value by key using square brackets `[]`.

```
```

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print(fruit['color'])
```

```
yellow
```

Adding items to a dictionary

Use square brackets `[]`.

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print(fruit['genus'])
print(fruit)
fruit['price'] = 0.10
print(fruit)
```

```
Musa
{'name': 'banana', 'color': 'yellow', 'genus': 'Musa'}
{'name': 'banana', 'color': 'yellow', 'genus': 'Musa', 'price': 0.1}
```

Using a dictionary as a collection of counters

Let's create a program that will help us visualize height distribution...

```
heights = [60, 61, 61, 67, 68, 70, 70, 70]

def histogram(data):
    frequencies = dict()
    for observation in data:
        if observation not in frequencies:
            frequencies[observation] = 1
        else:
            frequencies[observation] += 1
    return frequencies
```

```
print(histogram(heights))
```

```
{60: 1, 61: 2, 67: 1, 68: 1, 70: 3}
```

The dictionary's `get` method takes a key and a default value. If the key exists, `get` returns the corresponding value. Otherwise, it returns the default value.

How can we use the `get` method to simplify `histogram`?

```
heights = [60, 61, 61, 67, 68, 70, 70, 70]

# TODO: Simplify histogram
def histogram(data):
    frequencies = dict()
    for observation in data:
        if observation not in frequencies:
            frequencies[observation] = 1
        else:
            frequencies[observation] += 1
    return frequencies

print(histogram(heights))
```

```
{60: 1, 61: 2, 67: 1, 68: 1, 70: 3}
```

Let's see the histogram in action...

```
heights = []

def histogram(data):
    frequencies = dict()
    for observation in data:
        frequencies[observation] = frequencies.get(observation, 0) + 1
    return frequencies

distribution = histogram(heights)
for height in sorted(distribution):
    stars = '*' * distribution[height]
    print(f'{height} {stars}')
```

Traversing the keys of a dictionary

So far, we have looked at using a `for` loop to traverse the elements of a list...

```
fruits = ['apple', 'banana', 'pineapple']
for fruit in fruits:
    print(fruit)
```

```
apple
banana
pineapple
```

... the indices of a list...

```
fruits = ['apple', 'banana', 'pineapple']
for index in range(len(fruits)):
    print(index)
```

```
0
1
2
```

... and the characters of a string.

```
fruit = 'apple'
for letter in fruit:
    print(letter)
```

```
a
p
p
l
e
```

We can also use a `for` loop to traverse the keys of a dictionary.

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
```

```
for key in fruit:
    print(key)
```

```
name
color
genus
```

Let's modify the `for` loop so that it prints the values, too.

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
for key in fruit:
    print(key)
```

```
name
color
genus
```

Sorting the keys in a dictionary

Consider a dictionary that maps students to grades...

```
grades = {
    'lenny': 78,
    'barbara': 90,
    'george': 91,
    'amy': 94,
    'zack': 96
}
```

How can we print the key-value pairs in `grades` with the student names in alphabetical order?

You can sort the keys in a dictionary using the `sorted` function.

The `sorted` function takes a dictionary and returns the sorted keys.

```
grades = {
    'lenny': 78,
```

```
    'barbara': 90,
    'george': 91,
    'amy': 94,
    'zack': 96
}
print(sorted(grades))
for student in sorted(grades):
    print(f'{student} has a grade of {grades[student]}%.')
```

```
['amy', 'barbara', 'george', 'lenny', 'zack']
amy has a grade of 94%.
barbara has a grade of 90%.
george has a grade of 91%.
lenny has a grade of 78%.
zack has a grade of 96%.
```

Looking up a key by its value (reverse lookup)

We have seen that it is easy to look up a *value* by its *key*.

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print(fruit['name'])
```

```
banana
```

But, how can we look up a *key* by its *value*?

E.g. How can we look up what key corresponds to the value `banana`?

Looking up a key by its value is a more complicated process, called a **reverse lookup**.

The high-level procedure for a reverse lookup is

- For each key in a dictionary
    - Check the key's value
        - If the value matches the value you are looking for, return the key
- If you have checked the value of every key and none have matched the value you are looking for, raise a `LookupError`.

```
```

```
def reverse_lookup(d, value):
    for key in d:
        if d[key] == value:
            return key
    raise LookupError()
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print(reverse_lookup(fruit, 'banana'))
```

```
name
```

The `raise` statement

Exceptions occur when something "exceptionally" bad happens.

Types of exceptions we have encountered before

`KeyError`

```
fruit = {
    'name': 'banana',
    'color': 'yellow',
    'genus': 'Musa'
}
print(fruit['price'])
```

```
---------------------------------------------------------------------------
-

KeyError                                  Traceback (most recent call
last)

/var/folders/bj/bw1mwdzj6vsbs4zf676n7rw80000gq/T/ipykernel_73978/378237777
2.py in <module>
      4     'genus': 'Musa'
      5 }
----> 6 print(fruit['price'])


KeyError: 'price'
```

IndexError

```
fruits = ['apple', 'banana', 'pineapple']
print(fruits[3])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call
last)

/var/folders/bj/bw1mwdzj6vsbs4zf676n7rw80000gq/T/ipykernel_73978/154058710
9.py in <module>
      1 fruits = ['apple', 'banana', 'pineapple']
----> 2 print(fruits[3])


IndexError: list index out of range
```

FileNotFoundError

```
fin = open('does-not-exist.txt')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call
last)

/var/folders/bj/bw1mwdzj6vsbs4zf676n7rw80000gq/T/ipykernel_73978/38292061.
py in <module>
----> 1 fin = open('does-not-exist.txt')


FileNotFoundError: [Errno 2] No such file or directory: 'does-not-
exist.txt'
```

These `KeyError`, `IndexError`, and `FileNotFoundError` exceptions are raised internally by Python. We can raise our own exceptions by using the `raise` statement.

Use the keyword `raise` followed by the name of the exception you want to raise.

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

You should raise an exception when you want to let the user of your program or code know that something went wrong.

Other exceptions you could raise include:

- `Exception`
- `NotImplementedError`

You can find more exceptions that you can raise in the Python documentation.

Catching a raised exception

Remember that you should place risky code in a `try` statement. Risky code is any code that has a reasonable chance of causing an exception.

```
try:
    print('Before line that causes exception. Runs!')
    fin = open('does-not-exist.txt')
    print('After line that causes exception. Never runs!')
except:
    print('There was an exception.')
```

```
Before line that causes exception. Runs!
There was an exception.
```

Notice that as soon as the risky code causes an exception, the flow of execution jumps down to the `except` block. None of the code after the line that causes the exception is run.

```
try:
    print('Before line that causes exception. Runs!')
    raise Exception()
    print('After line that causes exception. Never runs!')
except:
    print('There was an exception.')
```

```
Before line that causes exception. Runs!
There was an exception.
```

Keys of a dictionary must be immutable

Keys can be strings, integers, floats, and even booleans! This is because they are immutable.

```
random = {
    'name': 'foobar',
    0: True,
    3.14: 'pie',
    False: 12345
}
```

Keys cannot be lists because they are mutable.

```
t = [1, 2, 3]
random[t] = 'list' # Causes an exception
```

```
---------------------------------------------------------------------------
-

NameError                                 Traceback (most recent call
last)

/var/folders/bj/bw1mwdzj6vsbs4zf676n7rw80000gq/T/ipykernel_73978/365068741
0.py in <module>
      1 t = [1, 2, 3]
----> 2 random[t] = 'list' # Causes an exception


NameError: name 'random' is not defined
```

Summary

- What dictionaries are and how to create them
- How to look up values by key
- How to add items to a dictionary
- How to check if something is a key or value in a dictionary
- How to use dictionaries to store counters
- How to traverse the keys of a dictionary
- How to sort the keys of a dictionary
- How to look up keys by value
- How to raise an exception

- What types keys can and cannot be

Office Hours

M 11:30 AM - 12:30 PM; 2:00 - 4:00 PM

Tu 10:00 AM - 1:00 PM

W 11:30 AM - 12:30 PM

F 11:30 AM - 12:30 PM; 3:00 - 4:00 PM