Previously:

Ordered collections:

- lists (flexible, mutable)
- tuples (simple, immutable)
- arrays (fixed-size, homogeneous, high-performance)

Unordered collections: sets

Today:

Dictionaries

- sophisticated *associative* collection
- semantics
- syntax
- usage

Finding things in collections

```
def find(needle, haystack):
    for value in haystack:
        if value == needle:
            print('Found it!')
            return

    print("Didn't find it.")

find('Jon', ['Alice', 'Bob', 'Charlie', 'Diana'])
find('Diana', ['Alice', 'Bob', 'Charlie', 'Diana'])
```

How long does this take?

How about in an array of 40 M elements? 40 B?

Naming things in collections

```
sid = int(input('Enter student ID> '))

index = None
for i, s in enumerate(something_that_returns_students()):
    if s.id == sid:
        index = i

print('Found student at index:', index)
```

What is the "name" for the student in this collection?

```
print(students[i])    # print student i
students[i].name()    # get student i's name
students[i] = ...     # set to another student
```

That's odd...

Does it matter whether you're at index 12 or 93?

- doesn't matter whether you registered first, tenth or last
- "Student in seat 4" not a meaningful way to refer to you!

What's a better way to refer to you?

- name
- student ID

Another type of collection?

Sometimes the *order* of things doesn't matter

Sometimes we need a sensible *name* for things

> *There are only two hard things in Computer Science: cache invalidation and naming things. — Phil Karlton*

Leon Bambrick
@secretGeek

There are 2 hard problems in computer science: cache invalidation, naming things, and off-by-1 errors.

Python dictionary

A *dictionary* holds *named* values

- Python type: `dict`
- it's an *unordered* collection
- every *value* in a dictionary also has a *key* (a name)
- can look up values **by key**
- can iterate over keys, values **or both**

Dictionary syntax

Create a dictionary:

```
students = {
    200125805: 'Jonathan Anderson',
    202412345: 'Somebody New',
}
```

- enclosed in curly *braces* (not brackets or parentheses)
- comma-separated *items*
- each item has a *key* and a *value*

Dictionary values

Can use any type for values:

```
csf_rooms = {
    2111: "Alice Faisal",
    2112: "Computer lab",
    # ...
    4101: "Workstations",
    4103: "VISOR lab",
    # ...
    4123: "Jonathan Anderson",
    # ...
}
```

```
course_averages = {
    1010: 57.5,
    1020: 67.6,
    1030: 78.7,
    1040: 74.9,
}
```

```
grades = {
    200125805: [90, 98],
    # ...
}
```

Dictionary keys

Can use *many* types for keys:

```
populations = {
    'CBS': 24_848,
    'Corner Brook': 19_886,
    'Gander': 11_054,
    'Grand Falls-Winsor': 13_725,
    'Mount Pearl': 24_284,
    'Paradise': 17_695,
    "St. John's": 106_172,
}
```

```
checkers = {
    (0, 0): 'red',
    (0, 2): 'red',
    # ...
    (7, 1): 'black',
    (7, 3): 'black',
}
```

```
an_error = {
    [0, 0]: 'whut',
}
```

Population data is a bit stale: it's as of the 2011 census.

Valid dictionary keys

`TypeError: unhashable type: 'list'` — ???

Keys must be *hashable*

Python's *immutable* containers are hashable

- tuples are OK (if its elements are hashable), lists are not
- strings are OK, arrays of characters are not

Indexing

Can access individual elements just like indexing:

```
s = students[200125805]          # looks a lot like a list or array
p = populations['Gander']        # well that's new!
populations["St. John's"] += 1   # congratulations to the new parents?
```

Iterating over dictionary keys and values

By default, you iterate over *keys*:

```
for sid in students:
    print(sid, ':', students[sid])

for city in populations.keys():         # this does the same thing
    print(city, ':', populations[city])
```

Can also iterate over values

```
for pop in populations.values():
    print(pop)     # but we don't know which city we're referring to
```

Iterating over dictionary items

Can also iterate over *items* (key, value tuples)

```
for sid, student in students.items():
    print(sid, ':', student)

for city, pop in populations.items():
    print(city, ':', pop)
```

Ordering *may not* be preserved*

** Fine print (not on the exam): Python 3.7+ preserves insertion order in the `dict` type, but many Python packages that interact with `dict` don't assume that ordering will be preserved, so they may not work to preserve it in the data you import or export.*

Depending on the version of Python and other factors, it's possible that iterating twice over a dictionary might give you a different order each time.

Using dictionaries

Helpful when *name* more important than *order*

Allow very fast search by key

- no need to look at all 40 B records!
- how? details will come later (ECE 4400 or equivalent)

Basis for lots of code in Python packages you may use

Example with pandas*

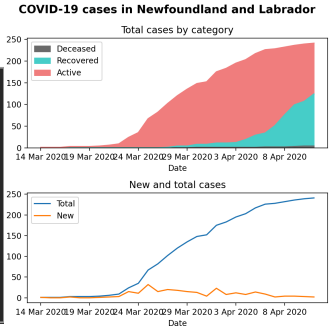
pandas frames behave like dicts:

```
import pandas as pd

#
# Read data and compute some summaries:
#
data = pd.read_csv('covid-data.csv')

data['Total'] = data['New'].cumsum()
data['Deceased'] = data['Deaths'].cumsum()
data['Recovered'] = data['Recoveries'].cumsum()

# ...
```



* See the rest of the code as well as the data it operates on

Summary:

Dictionaries

- sophisticated *associative* collection
- semantics
- syntax
- usage