Dictionaries (Type `dict`)

**Description**
- List of key-value pairs
  - Keys are unique
  - Values need not be
- Example: `net-ids`
  - `net-ids` are **unique** (a key)
  - Names need not be (values)
  - `js1` is John Smith (class '13)
  - `js2` is John Smith (class '16)
- Many other applications

**Python Syntax**
- Create with format: `{k1:v1, k2:v2, ...}`
- Keys must be non-mutable
  - Ints, floats, bools, strings
  - Not lists or custom objects
- Values can be anything

**Example:**
```
d = {'js1': 'John Smith', 'js2': 'John Smith', 'wmw2': 'Walker White'}
```

Using Dictionaries (Type `dict`)

- Access elts. like a list
  - `d['js1']` evaluates to `John`
  - But cannot slice ranges!
- Dictionaries are **mutable**
  - Can reassign values
  - `d['js1'] = 'Jane'`
  - Can add new keys
  - `d['aa1'] = 'Allen'`
  - Can delete keys
  - `del d['wmw2']`

Dictionaries and For-Loops

- Dictionaries != sequences
  - Cannot slice them
- **Different** inside for loop
  - Loop variable gets the key
  - Then use key to get value
- Can **extract iterators** with dictionary methods
  - Key iterator: `d.keys()`
  - Value iterator: `d.values()`
  - Key-value pairs: `d.items()`

```python
for k in d:
    # Loops over keys
    print(k)  # key
    print(d[k])  # value

# To loop over values only
for v in d.values():
    print(v)  # value
```

See `grades.py`

Thinking About Assignment 2

- **A2:** three color models
  - RGB: 3 ints 0 to 255
  - CMYK: 4 floats 0.0 to 100.0
  - HSV: 3 floats, mult. bounds
  - We could represent as lists
- Can get really confusing
  - Easy to mix-up models
  - Easy to go out of bounds
- **We want custom types**
  - One for each color model
  - Motivation for `class`

Classes are Customized Types

- Classes are how we add new types to Python
- Values look like `dicts`
  - Represent as a folder
  - Variables are named

```python
id2 = RGB('red', 255, 'green', 128, 'blue', 0)
```

Why Are They Better Than `dicts`?

- Can add new variables
- Does not check bounds of the content variables
- Variables fixed (sort-of)
- Possibly checks bounds of the content variables
Using Classes in Python

• **Modules** provide classes
  * Import to use the class
  * Will show contents later

• **Example:** `cornell`
  * Color classes for A2: RGB, CMYK, HSV
  * Geometry classes: `Point2`, `Point3`
  * Will make our own later

---

Constructors: Function to make Objects

• How do we create objects?
  * Other types have **literals**
    * Example: 1, 'abc', `true`
  * No such thing for objects

• **Constructor Function:**
  * Same name as the class
    * Example: `Point3(0,0,0)`
  * Makes an object (manila folder)
  * Returns folder ID as value

• Example: `p = Point3(0, 0, 0)`
  * Creates a Point object
  * Stores object’s ID in `p`

---

Constructors and Modules

```python
>>> import cornell
>>> p = cornell.Point3(0,0,0)
```

• Need to import module that has Point class.

```python
>>> id(p)
```

• Shows the ID of `p`.

---

Object Variables

• Variable stores object name
  * **Reference** to the object
  * Reason for folder analogy

• Assignment uses object name
  * Example: `q = p`
  * Takes name from `p`
  * Puts the name in `q`
  * Does not make new folder!

• **Like we saw with lists**
  * Reason for using folders

---

Objects and Attributes

• Attributes are variables that live inside of objects
  * Can use in expressions
  * Can assign values to them

• Access: `<variable>`, `<attr>`
  * Example: `p.x`
  * Look like module variables

• Putting it all together
  * `p = cornell.Point3(1,2,3)`
  * `p.x = p.y + p.z`

---

Methods: Functions Tied to Objects

• **Method:** function tied to object
  * Method call looks like a function call preceded by a variable name:
    `(variable).<method>(<arguments>)`

  • Example: `p.distance(q)`
  * Example: `p.abs()` # makes `x,y,z ≥ 0`

• Just like we saw for strings
  * `s = 'abracadabra'`
  * `s.index('a')`

• Are strings objects?