Processing Lists: builtins

- `sum(x)` adds up all the elements in the list `x`
  - They must all be numbers!
- `min(x)` or `max(x)` find the min/max value in list `x`
  - They use the same ordering as `sort()`
- `range(a, b, c)` produces `[a, a+c, a+2*c, …, a+c*(b-a)/c]`
  - Starts at `a`, increases by `c` each time, until `b` (or less)
  - The argument `c` is optional; `c = 1` by default
- `list(x)` converts `x` (such as a string) to a list
  - Example: `list('mimsy')` produces `['m', 'i', 'm', 's', 'y']`

The Map Function

- `map(function, list)`
  - Function has to have exactly 1 parameter
  - Otherwise, get an error
  - Returns a new list
- `map(f, x)`
  - Does the same thing as:
    - `def map(f, x):
      result = []
      for y in x:
        result.append(f(y))
      return result`

Overview of Two-Dimensional Lists

- Access value at row 3, col 2:
  - `d[3][2]`
- Assign value at row 3, col 2:
  - `d[3][2] = 8`
- An odd symmetry
  - Number of rows of `d`: `len(d)`
  - Number of cols in row `r` of `d`: `len(d[r])`

How Multidimensional Lists are Stored

- `b = [[9, 6, 4], [5, 7, 7]]`
  - `b` holds name of a one-dimensional list
    - Has `len(b)` elements
    - Its elements are (the names of) 1D lists
  - `b[i]` holds the name of a one-dimensional list of ints
    - Has `len(b[i])` elements

Image Data: 2D Lists of Pixels

- `b[0][0]` is a white pixel
  - RGB values:
Ragged Lists: Rows w/ Different Length

- \( b = \begin{bmatrix} [17,13,19], [28,95] \end{bmatrix} \)
- Will see applications of this later

Slices and Multidimensional Lists

- Only “top-level” list is copied.
- Contents of the list are not altered
- \( b = \begin{bmatrix} [0, 6], [4, 6], [7, 7] \end{bmatrix} \)

Dictionaries (Type \texttt{dict})

- 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

- Create with format: \{\( k1: v1, k2:v2, \ldots \)\}
  - Keys must be non-mutable
    - ints, floats, bools, strings
  - Not lists or custom objects
  - Values can be anything
  - Example: \( d = \{\text{'js1': 'John'}, \text{'js2': 'John'}, \text{'wmw2': 'Walker White'}\} \)

Using Dictionaries (Type \texttt{dict})

- Access elts. like a list
  - \( d[\text{'js1'] \) evaluates to ‘John’
  - But cannot slice ranges!
- Dictionaries are mutable
  - Can reassign values
  - \( d[\text{'js1']] = \text{'Jane'} \)
  - Can add new keys
  - \( d[\text{'aa1']] = \text{'Allen'} \)
  - Can delete keys
    - \( \text{del } d[\text{'wmw2']} \)

Using Dictionaries (Type \texttt{dict})

- Access elts. like a list
  - \( d[\text{'js1'] \) evaluates to ‘John’
  - But cannot slice ranges!
- Dictionaries are mutable
  - Can reassign values
  - \( d[\text{'js1']] = \text{'Jane'} \)
  - Can add new keys
  - \( d[\text{'aa1']] = \text{'Allen'} \)
  - Can delete keys
    - \( \text{del } d[\text{'wmw2']} \)

- Key-Value order in folder is not important

Using Dictionaries (Type \texttt{dict})

- Access elts. like a list
  - \( d[\text{'js1'] \) evaluates to ‘John’
  - But cannot slice ranges!
- Dictionaries are mutable
  - Can reassign values
  - \( d[\text{'js1']] = \text{'Jane'} \)
  - Can add new keys
  - \( d[\text{'aa1']] = \text{'Allen'} \)
  - Can delete keys
  - \( \text{del } d[\text{'wmw2']} \)

Using Dictionaries (Type \texttt{dict})

- Access elts. like a list
  - \( d[\text{'js1'] \) evaluates to ‘John’
  - But cannot slice ranges!
- Dictionaries are mutable
  - Can reassign values
  - \( d[\text{'js1']] = \text{'Jane'} \)
  - Can add new keys
  - \( d[\text{'aa1']] = \text{'Allen'} \)
  - Can delete keys
  - \( \text{del } d[\text{'wmw2']} \)

Dictionaries and For-Loops

- Dictionaries != sequences
  - Cannot slice them
- Different inside for loop
  - Loop variable gets the key
  - Then use key to get value
- Has methods to convert dictionary to a sequence
  - Seq of keys: \( d.keys() \)
  - Seq of values: \( d.values() \)
  - key-value pairs: \( d.items() \)

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 \texttt{grades.py}