

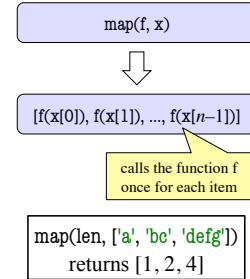
Processing Lists: builtins

- `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
- Does the same thing as


```
def map(f,x):
    result = [] # empty list
    for y in x:
        result.append(f(y))
    return result
```



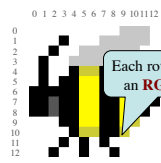
Two Dimensional Lists

Table of Data

	0	1	2	3
0	5	4	7	3
1	4	8	9	7
2	5	1	2	3
3	4	1	2	9
4	6	7	8	0

Each row, col has a value

Images



Store them as lists of lists (**row-major order**)
`d = [[5,4,7,3],[4,8,9,7],[5,1,2,3],[4,1,2,9],[6,7,8,0]]`

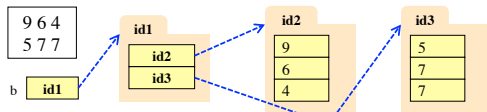
Overview of Two-Dimensional Lists

- 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])`

	0	1	2	3
d 0	5	4	7	3
1	4	8	9	7
2	5	1	2	3
3	4	1	2	9
4	6	7	8	0

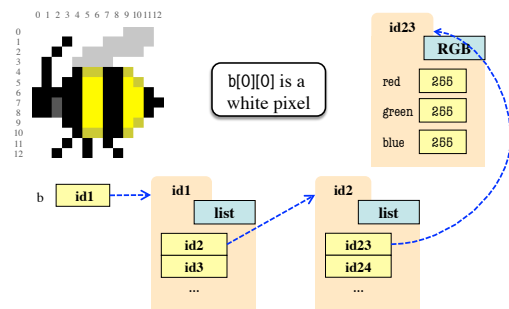
How Multidimensional Lists are Stored

- `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



Ragged Lists: Rows w/ Different Length

- `b = [[17,13,10],[28,95]]`

- Will see applications of this later

Slices and Multidimensional Lists

- Only "top-level" list is copied.
- Contents of the list are not altered

`b = [[0, 6], [4, 8], [7, 7]]`

`x = b[:2]`

Dictionaries (Type dict)

Description	Python Syntax
<ul style="list-style-type: none"> List of key-value pairs <ul style="list-style-type: none"> Keys are unique Values need not be Example: net-ids <ul style="list-style-type: none"> 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 	<ul style="list-style-type: none"> Create with format: <code>{k1:v1, k2:v2, ...}</code> Keys must be non-mutable <ul style="list-style-type: none"> ints, floats, bools, strings Not lists or custom objects Values can be anything Example: <pre>d = {'js1':'John Smith', 'js2':'John Smith', 'wmw2':'Walker White'}</pre>

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']`

```
d = {'js1':'John','js2':'John',
     'wmw2':'Walker'}
```

Key-Value order in folder is not important

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']`

```
d = {'js1':'John','js2':'John',
     'wmw2':'Walker'}
```

Deleting key deletes both

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 grades.py