Strings, Lists — Collections — Sequence

Strings: immutable; Lists: mutable

Tuple – immutable, sequence

```
(2, "hello", True)
p = (3, 2.5, 6.7)      # 3D point
```

"Comment"

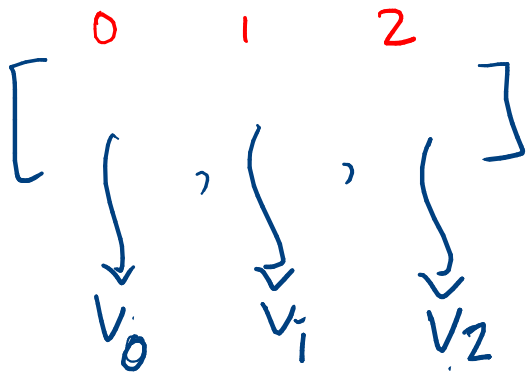
```
xcoord = p[0]
ycoord = p[1]
zcoord = p[2]
```

```
p[0] = 7.5   X      # Immutable!
```

```
p = (7.5, p[1], p[2])
```

List, more abstractly:

Map from positions to values



Generalize this:

Assign "keys" to values

$f : \{k_1 \mapsto v_1, k_2 \mapsto v_2, k_3 \to v_3\}$

Dictionary

Keys - any immutable value

Empty dictionary: {}

```
d = {}
d[5] = "hello"
d[True] = 7
d[(6,5)] = 9.5
d[5] = 26
```

5 ↦ ~~"hello"~~ 26,
True ↦ 7
(6,5) ↦ 9.5

Mutable

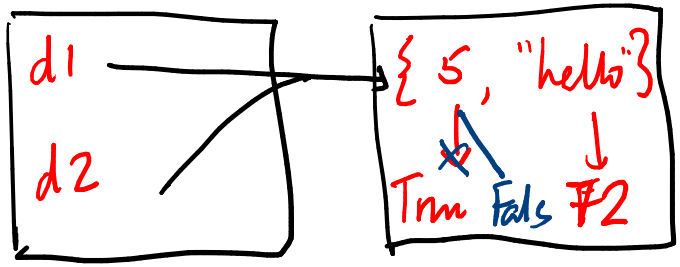
Set up a dictionary directly

```
d = {k1:v1, k2:v2, k3:v3}
```

```
d = {}
d[k1] = v1
d[k2] = v2
d[k3] = v3
```

```
d1 = {5:True, "hello":72}
d2 = d1
d2[5] = False
```

d1[5] also False



Example: Given a list of words, compute frequency of each word

Maintain a dictionary - each word is a key

Increment value for key k whenever k occurs

```
count = {}
.
.
.
count[w] = count[w] + 1
```

What happens when w appears for the first time?

count[w] = 1

For lists:

v in l membership test

Dictionary

Extract keys & values as lists

d.keys() (almost) a list of keys,
d.values() values

∴ if (w in count.keys()):
    count[w] = count[w]+1
else
    count[w] = 1

Order of d.keys() is "random"

Internally organized like a search tree to optimize looking up d[k]

Sorting a list:

l.sort() — sorts l in place

sorted(l) — returns a sorted copy of l, l is unchanged

To process a dictionary systematically

for k in sorted(d.keys()): ← requires all keys to be comparable

Structure of a Python program

```
def f(..):
    ≡
```

In principle, can have executable code here

```
def g(..):
    ≡
```

→ function defn is like g = ≡ } fn defn

≡ } code to execute — implicit function run first

Functions must be defined before used

Mutual recursion allowed

"Higher order" functions are allowed

- ie. can pass functions as arguments

map(f, l) $l = [v_0, v_1, \ldots, v_{n-1}]$

f ↓ ↓ ↓

not quite a list (like range(), d.keys() etc)

$[f(v_0), f(v_1), \ldots, f(v_{n-1})]$

filter(p, l) p : Values → {True, False}

... filter(iseven, l) ...

List comprehension

Haskell: `l = [ f x | x <- ... , p x ]`

map generate filter

In Python

```
[ x * x for x in range(10) if x%2 == 0 ]
```

sqr(x) if iseven(x)

Recall

```
zerolist = [0,0,0]
zeromatrix = [zerolist, zerolist, zerolist]
```

Not good, zerolist is shared

Instead

```
[[0 for y in range(3)] for x in range(3)]
```

Functions with optional arguments

int("3") ⇝ 3

int("a") ⇝ error

int("a", 16) ⇝ 10

↳ optional, default is 10

```
def int(x, b=10):
    ...
```

↖ All optional arguments to right

```
def f(a, b, c=6, d=10):
    ...
```

f(3, 5, 8)

↙

a = 3

b = 5

c = 8

d = 10

f(a=3, d=7, b=3)

- use argument names explicitly, order is flexible

```
def f(--):
    :
    x = g(--)

def g(--):
    :
    y = f(--)

    :

z = f(7)
```

Can use f, but not g,
Using f will generate error

For "clean" code

```
def f1():
    ≡
def f2():
    ≡
    :
def main():
    ≡

main()
```

← execute here