Lecture 7, 11 October 2021

Mutable and immutable values

- Property of values, not names/variable
- Mutability - can the value being pointed to change in place?
- There is only "one copy" of every immutable value
- If we update `x` from 3 to 4, `x` points to the one value of 4 instead of the one value of 3

```
x = 3
y = x
x is y, x == y
```

```
(True, True)
```

```
x·=·3
y·=·x
x = 4
x is y, x == y
```

```
(False, False)
```

- Even in a list, you have to interpret mutability correctly
- If `l[i]` points to an immutable value and you make another name `x` point to the same value, updating `l[i]` will not update `x`
- Only when one list is aliased to another will updating `l[i]` in the first list also reflect in the same position in the second list

```
l = [7,8]
l2 = l
x = l[0]  # l[0] is now like an individual variable y
l[0] = 9
x is l[0], x == l[0], l[0] is l2[0]
```

```
(False, False, True)
```

Dealing with immutability

- `a,b = b,a` exchanges the values of `a` and `b`
- Can we write a function `swap(a,b)` that exchanges the values of its (immutable) arguments?
- Can only do something like `a,b = swap(a,b)`

Dictionaries

- A list is a collection indexed by position

- A list can be thought of as a function $f : \{0, 1, \ldots, n-1\} \to \{v_0, v_1, \ldots, v_{n-1}\}$
  - A list maps positions to values
- Generalize this to a function $f : \{x_0, x_1, \ldots, x_{n-1}\} \to \{v_0, v_1, \ldots, v_{n-1}\}$
  - Instead of positions, index by an abstract *key*
- **dictionary**: maps keys, rather than positions, to values
- Notation:
  - `d = {k1:v1, k2:v2}`, enumerate a dictionary explicitly
  - `d[k1]`, value in dictionary `d1` corresponding to key `k1`
  - `{}`, empty dictionary (`[]` for lists, `()` for tuples)
  - Keys must be immutable values

```
names = ["Abha","Bunty"]
#bdays = {}
bdays = {"Abha":"03-05-2001", "Bunty":"17-10-1999"}
```

```
bdays["Bunty"]
```

```
'17-10-1999'
```

- Accessing a non-existent key results in `KeyError`
- Analogous to `IndexError` for invalid position in a list

```
bdays["Chitra"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-9-0ddd3b92360d> in <module>()
----> 1 bdays["Chitra"]

KeyError: 'Chitra'
```

SEARCH STACK OVERFLOW

```
l = [0,1,2]
l[3] = 3
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-10-dd0d4465d29b> in <module>()
      1 l = [0,1,2]
----> 2 l[3] = 3

IndexError: list assignment index out of range
```

SEARCH STACK OVERFLOW

- Assigning to non-existent key creates a new key-value pair
  - Unlike lists, where we cannot create a new position by assigning outside the current list

- `d[k] = v` creates `k` if it does not exist, updates the value at `d[k]` if it does exist

```
bdays["Chitra"] = "13-08-2000"
bdays
```

```
{'Abha': '03-05-2001', 'Bunty': '17-10-1999', 'Chitra': '13-08-2000'}
```

```
bdays["Chitra"] = "13-09-2000"
bdays
```

```
{'Abha': '03-05-2001', 'Bunty': '17-10-1999', 'Chitra': '13-09-2000'}
```

- Any immutable value can be a key
- No requirement that all keys (or values) have a uniform type

```
bdays[0] = 7
bdays
```

```
{0: 7, 'Abha': '03-05-2001', 'Bunty': '17-10-1999', 'Chitra': '13-09-2000'}
```

- Key must be an immutable value
    - List cannot be used as a key

```
bdays[[1]] = 77b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-f76a5cf6ed76> in <module>()
----> 1 bdays[[1]] = 77

TypeError: unhashable type: 'list'
```

SEARCH STACK OVERFLOW

- Value at a key can be a list
- Use multiple subscripting to access inner components
- Similar to nested lists

```
bdays[1] = [77,88]
bdays, bdays[1][1]
```

```
({0: 7,
  1: [77, 88],
  'Abha': '03-05-2001',
  'Bunty': '17-10-1999',
  'Chitra': '13-09-2000'},
 88)
```

- Likewise, value can be a nested dictionary

```
brothers = {}
brothers["ahmed"] = {"first":"abdul", "second":"salman"}
brothers, brothers["ahmed"]["second"]
```

```
({'ahmed': {'first': 'abdul', 'second': 'salman'}}, 'salman')
```

Operating on dictionaries

- How do we run through all entries in a dictionary - the equivalent of `for x in l`?
- `d.keys()`, `d.values()` generate sequences corresponding to the keys and values of `d`, respectively
- Like `range()` these are not directly lists, use `list(d.keys())` if you want a list

```
bdays.keys()  # Not quite a list, a bit like range()
```

```
dict_keys(['Abha', 'Bunty', 'Chitra', 0, 1])
```

```
for name in bdays.keys():
  print(name)
```

```
Abha
Bunty
Chitra
0
1
```

```
namelist = list(bdays.keys())
namelist
```

```
['Abha', 'Bunty', 'Chitra', 0, 1]
```

```
list(bdays.values())  # The values in a dictionary
```

```
['03-05-2001', '17-10-1999', '13-09-2000', 7, [77, 88]]
```

- In what order does `d.keys()` list the keys?
- In theory, this order is arbitrary and you should not make any assumptions
- In practice, from some recent version of Python (3.6?) keys are listed in the order added
- If dictionary keys are of the same type, use `sorted(d.keys())` to get them in sorted

```
d = {}
d['a'] = 7
d['c'] = 9
d['b'] = 8
```

```
list(d.keys()), sorted(d.keys())
```

```
(['a', 'c', 'b'], ['a', 'b', 'c'])
```

```
for name in sorted(d.keys()):
  print(d[name])
```

Accumulating values

- We have a list of pairs (name,marks) of marks in assignments of students in a course
- We want to report the total marks of each student
- Create a dictionary `total` whose keys are names and whose values are total marks for that name
- How would we do this?

```
marklist = [("abha",75),("bunty",58),("abha",86),("chitra",77),("bun
total = {}
for markpair in marklist:
  name = markpair[0]
  marks = markpair[1]
  # add marks to total[name], only if tota[name] already exist, othe
  if name in total.keys():  # check if a key exists already
    total[name] = total[name] + marks
  else:



    total[name] = marks
print(total)

```

```
{'abha': 161, 'bunty': 150, 'chitra': 77}
```

Representing sets

- Maintain a set X (from a universe U)
- Digression on set theory as a foundation for mathematics
  - 0 is $\emptyset$, 1 is $\{0\} = \{\emptyset\}$, 2 is $\{0,1\} = \{\emptyset, \{\emptyset\}\}, \ldots, n = \{1, 2, \ldots, n-1\}$
  - Build up arithmetic and all of mathematics from these foundations
  - Is every collection a set?
  - What about the set of all sets?
  - Some sets contain other sets, some do not
    - For instance, the powerset of X consists of a subsets of X
  - Bertrand Russell: Let X be the set of all sets that do not contain themselves
  - Does X contain X?
    - If X does not contain X, X must contain X by its definition
    - Paradox!
  - Similar to saying "I am a liar"
    - Should you believe me?
    - If you do believe me, you should not!

  - Russell's paradox tells us the collection of all sets is not a set
  - Can only build sets from existing sets, not create arbitray collections and call them sets
- Representing sets using functions
  - A subset $X \subseteq U$ is the same as a function $X : U \rightarrow \{\text{True}, \text{False}\}$
  - Say, $U = \{0, 1, \ldots, 999\}$, P = primes in U
  - $P = \{2, 3, 5, 7, \ldots, 997\}$
  - $P : \{0, 1, \ldots, 999\} \rightarrow \{\text{True}, \text{False}\}$
- Create a dictionary whose keys are those values x for which $P(x) = \text{True}$
  - `primes = {}`
  - `primes[2] = True`
  - `primes[3] = True`
  - ...
  - `primes[997] = True`
- The set is implicitly the collection of keys of the dictionary
  - Can also explicitly add `primes[0] = False`, `primes[1] = False`, ..., but this is redundant
- **Exercise**: If `d1` and `d2` both represent sets over U, how do we compute `d1` $\cup$ `d2`, `d1` $\cap$ `d2`, $U\setminus$ `d1` (complement of `d1` wrt U)?