

# Py ch10: Dictionaries

---

9 Nov 2007

CMPT14x

Dr. Sean Ho

Trinity Western University

# Quiz07 (Py ch15)

- Contrast **procedural** vs. **object-oriented** programming.
- Define: **object**, **class**, **instance**, **attribute**, **method**
- Come up with an **application** for records, and **define** (in pseudocode) a suitable **record type**. Include the **type** of each record field.

# Addendum on class variables

- Two kinds of **attributes** of an object:
  - **Class** attributes: **shared** by all instances

**class Fraction:**

**numer = 0**

**denom = 1**

- **Instance** attributes: specific to **this** instance

**class Fraction:**

**def \_\_init\_\_(self, n=0, d=1):**

**self.numer = n**

**self.denom = d**

- For **records**, it's actually better to use **instance** attributes, not class attributes

# What's on for today (Py ch10)

## ■ Dictionaries

- Keys and values
- Basic dictionary methods:
  - ◆ .keys(), .values(), .items()
- Iterating through dictionaries
- Other dictionary methods:
  - ◆ len(), del, in, .get(), .copy()
- Application: hinting
  - ◆ Fibonacci example

# Python type hierarchy (partial)

- Atomic types

- Numbers

- ◆ Integers (int, long, bool): 5, 500000L, True
    - ◆ Reals (float) (only double-precision): 5.0
    - ◆ Complex numbers (complex): 5+2j

- Container (aggregate) types

- Immutable sequences

- ◆ Strings (str): "Hello"
    - ◆ Tuples (tuple): (2, 5.0, "hi")

- Mutable sequences

- ◆ Lists (list): [2, 5.0, "hi"]

- Mappings

- ◆ Dictionaries (dict): {"apple": 5, "orange": 8}

# Dictionaries

- Python **dictionaries** are mutable, unsorted containers holding associative **key-value** pairs
- **Create** a dictionary with curly braces **{}**:
  - ◆ **appleInv = {'Fuji': 10, 'Gala': 5, 'Spartan': 7}**
- **Index** a dictionary using a **key**:
  - ◆ **appleInv['Fuji']**            **# returns 10**
- **Values** can be any object; need not be same **type**:
  - ◆ **appleInv['Rome'] = range(3)**
- **Keys** can be any **immutable** type:
  - ◆ **appleInv[('BC', 'Red Delicious')] = 12**

# Dictionaries: `keys()` and `values()`

- All dictionaries have the following **methods**:
  - **`keys()`**: returns a list of all the **keys**
    - ◆ `appleInv.keys()`  
['Fuji', 'Spartan', 'Rome', 'Gala', ('BC', 'Red Delicious')]
  - **`values()`**: returns a list of all the **values**
    - ◆ `appleInv.values()`  
[10, 7, [0, 1, 2], 5, 12]
- Dictionaries are **unsorted**!
  - The order of `keys()` and `values()` will correspond if the dictionary isn't modified

# Iterating through dictionaries

- To print our apple inventory:
  - ◆ `for appleType in appleInv.keys():`
    - `print "We have", appleInv[appleType], \`
      - `appleType, "apples."`
- Output:
  - ◆ We have 10 Fuji apples.
  - ◆ We have 7 Spartan apples.
  - ◆ We have [0, 1, 2] Rome apples.
  - ◆ We have 5 Gala apples.
  - ◆ We have 12 ('BC', 'Red Delicious') apples.



# Other dictionary methods

- `len(appleInv)`
- `del appleInv['Fuji']`
- `'Fuji' in appleInv`
- `appleInv.get('Braeburn', 0)`
  - Return **default** value if key is not in dictionary
- `appleInv.items()`
  - Returns a copy of the dictionary as a **list** of (**key**, **value**) **tuples**
- `appleInv.copy()`
  - **Shallow** copy

# Dictionary application: hinting

- Py ch10 illustrates a cool use of **dictionaries**:
- **Hinting**: save (cache) previously-calculated values for future use
- **Fibonacci** example:

```
def fib(n):
```

```
    if n == 0 or n == 1:
```

```
        return 1
```

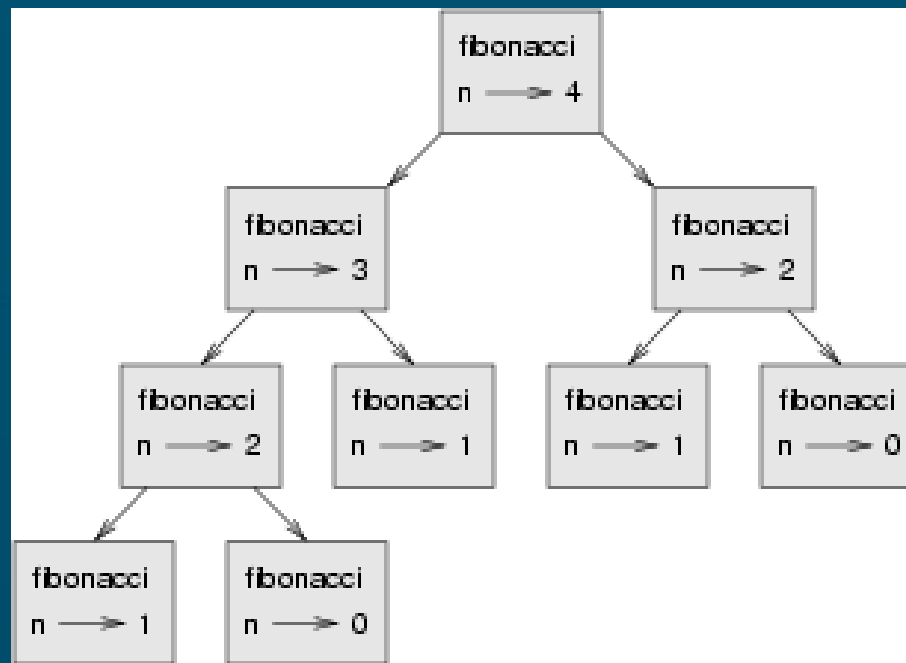
```
    return fib(n-1) + fib(n-2)
```

- But this is very slow and **inefficient**!
  - Try **fib(28)**, **fib(29)**, **fib(30)**, ....

- Fibonacci numbers get very **big** very fast

# Fibonacci revisited

- The **call-graph** for **fib()** shows that, e.g, **fib(2)** gets **recalculated** many times:



$O(n^2)$  calls in  
the graph

- If we **save** the value of **fib(2)** the first time it's calculated, we can **reuse** that **hint**

# Hinting Fibonacci

- Use a **dictionary** to store the precalculated **hints**:
  - **Key** is **n**; **value** is **fib(n)**
  - When we calculate a **fib()**, **add** it to the dict
  - Before calculating a **fib()**, **check** to see if it's already in the dictionary of **hints**

```
fibHints = {0:1, 1:1}
```

```
def hFib(n):
```

```
    if n in fibHints.keys():
```

```
        return fibHints[n]
```

```
    fibHints[n] = hFib(n-2) + hFib(n-1)
```

```
    return fibHints[n]
```

# Review of today (Py ch10)

## ■ Dictionaries

- Keys and values
- Basic dictionary methods:
  - ◆ `.keys()`, `.values()`, `.items()`
- Iterating through dictionaries
- Other dictionary methods:
  - ◆ `len()`, `del`, `in`, `.get()`, `.copy()`
- Application: hinting
  - ◆ Fibonacci example