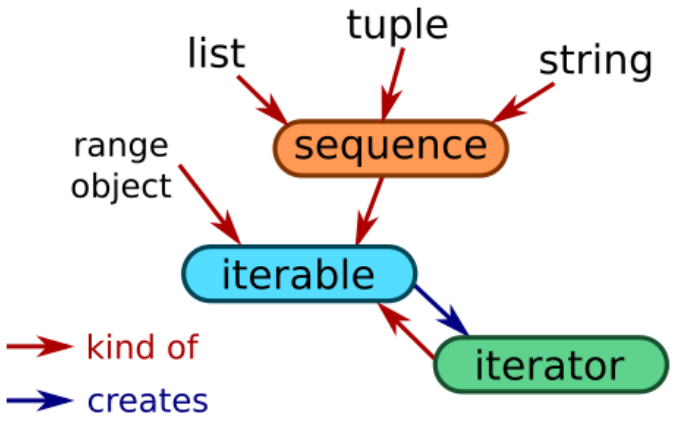
Computational Thinking

Lecture 08b: Iterable and Set

University of Engineering and Technology
VIETNAM NATIONAL UNIVERSITY HANOI

Outline

- Iterable: why iteration works with dictionaries
- Set: another built-in data type



Iterable

For-loops Over Dictionaries

Syntax

```
for <var> in <dict>:
    <statement>
    …
    <statement>
```

Example

```
for key in dct:
    print(key)
    print(dct[key])
```

Execution (simplified):

- Assign the first key of <dict> to <var>. Execute all the indented statements.
- Assign the second key of <dict> to <var>. Execute all the indented statements again.
- …
- Assign the last key of <dict> to <var>. Execute all the indented statements one last time.

Why Do For-loops Work with Dictionaries?

- Previously we learned that for-loops work over **sequences**
- But dictionaries do not support numerical indexes like d[0]
- So dictionaries are not sequences...

What Can We Loop Over?

- So far, we have always looped over sequence types
- **Sequences** have two operations: len() and indexing
  - You **can** always know in advance how many items there are
  - You **can** always skip around and get to items in any order
- But for-loops can also loop over a more general type: **iterable**
- An **iterable** type supports the iter() operation, which returns an iterator.
- An **iterator** has just one operation: next()
  - You **cannot** determine in advance how many items an iterator will produce
  - You **cannot** skip around; have to get them in whatever order the iterator chooses

Lists are Iterable

```
>>> lst = [1,2]
>>> lst_iterator = iter(lst)
>>> next(lst_iterator)
1
>>> next(lst_iterator)
2
>>> next(lst_iterator)
StopIteration
```

Dictionary keys are returned by iterator **in order of insertion** — not any kind of numerical or sorted order

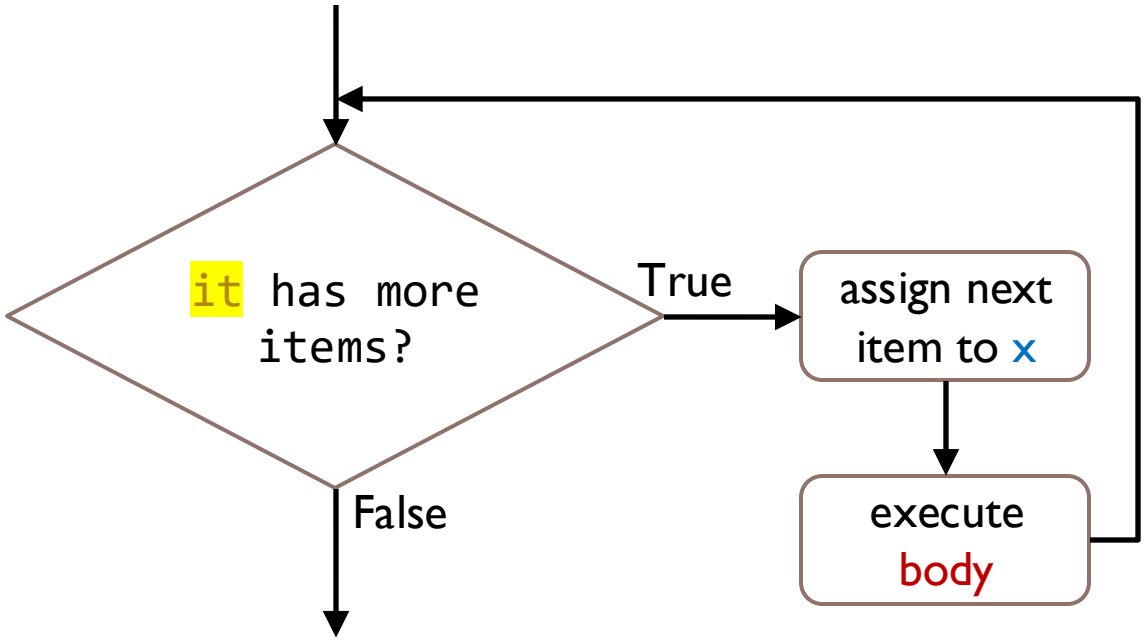
Dictionaries are Iterable

```
>>> dct = {'b':2, 'a':1}
>>> d_iterator = iter(dct)
>>> next(d_iterator)
'b'
>>> next(d_iterator)
'a'
>>> next(d_iterator)
StopIteration
```

Full Explanation: Execution of For-loop

```
for x in seq it:
    body
```

This diagram is another **flowchart**



The check for "more items?" happens n+1 times if there are n items. The last check is when the StopIteration occurs.

Dict. Methods that Return Iterables (1)

d.keys(): an iterable of the **keys** of d

Want the keys as a list? Easy:
list(d.keys())

Want to loop over keys? Easy, either of these works:

```
for key in d:
    # body

for key in d.keys(): # needlessly complicated
    # body
```

Dict. Methods that Return Iterables (2)

d.values(): an iterable of the **values** of d

Want the values as a list? Easy:
list(d.keys())

Want to loop over values ? Easy, either of these works:

```
for val in d.values():
    # body

for key in d:
   val = d[key]
    # rest of body
```

Dict. Methods that Return Iterables (3)

d.items(): an iterable of the **items** of d

Each item is a tuple of a key and value

Want to loop over items? Easy:

```
for (key, val) in d.items():
    # body
```

Back to: How to Print Character Count

```
def print_counts(counts):
    """Print a horizontal bar chart showing the
    counts, a dictionary mapping characters to
    integers."""
    for char in counts:
        count = counts[char]
        print(char + ' ' + '*' * count)
```

```
>>> print_counts(char_counts('abba!'))
a **
b **
! *
```

Set

Data Collections

- A **sequence** is a collection of data
  - Supports the **in** operation
  - Supports **lookup** of value by position, i.e., indexing
  - Examples: **list** & **tuple**
- A **dictionary** is a collection of data
  - Supports the **in** operation (on keys)
  - Supports **lookup** of value by key
- A **set** is also a collection of data
  - Supports the **in** operation
  - Like the mathematical notion of a **set**
  - Like a **list**, except there's no notion of position/order or multiple copies of an item
  - Like a **dictionary**, except there's only keys — no notion of looking up extra data value associated with keys

Sets

```
>>> s = {1, 2, 3}
>>> 1 in s
True
>>> 0 in s
False
>>> s.add(0)
>>> s
{0, 1, 2, 3}
>>> 0 in s
True
>>> s.add(1)
>>> s
{0, 1, 2, 3}
```

Curly braces, like dictionaries (and math)

Not a list: each item is unique; multiple copies not possible

```
>>> s.remove(0)
>>> s
{1, 2, 3}
>>> s.add(1024)
>>> s
{1024, 1, 2, 3}
>>> s.add(0)
{1024, 1, 2, 3, 0}
```

Not a list: no notion of order,* even if left side of this slide makes it seem that way.

How to Print the Items of a Set?

Easy, sets are also **iterable**

```
for item in s:
    # body
```

```
def print_set(s):
    for item in s:
        print(item)
```

What are the Unique Characters in String?

```
def uniq_chars(s):
    """Return a set containing the unique
    characters of string s."""
    uniq = set() # create empty set
    for char in s:
        uniq.add(char)
    return uniq
```

```
>>> uniq_chars('couscous')
{'o', 'u', 's', 'c'}
```

Set Constructor Function

Can construct a set out of any sequence:

```
# easier solution to problem
# on previous slide
>>> set('couscous')
{'o', 'u', 's', 'c'}

>>> set([1,1,2,3])
{1, 2, 3}
```