Dictionaries in Python

Sarah Khoja

Last time

- Data types:
  - Primitive: int, float, bool…
  - Sequences: string, list, tuple
- Now:
  - Mappings:
    - Dictionaries- a mutable object type

Necessity of Dictionaries

- What if you have multiple lists, but they contain related data.
  - Example:

```
name=["Kirk", "Spock", "Scotty"]
salary=[25, 30, 35]
job=["Captain", "Vulcan", "Fixer Guy"]
```

Necessity of Dictionaries

- Three lists
- Maintain size of lists
  - Must be equal!
- Storing the related data at the correct index

```
name=["Kirk", "Spock", "Scotty"]
salary=[25, 30, 35]
job=["Captain", "Vulcan", "Fixer Guy"]
```

Necessity of Dictionaries

- Imagine the code for updating or retrieving the student info:

```
def get_info(emp, name_list, sal_list, job_list):
  i= name_list.index(emp)
  s=sal_list[i]
  j= job_list[i]
  return (s, j)
```

Necessity of Dictionaries

```
name=["Kirk", "Spock", "Scotty"]
salary=[35, 30, 25]
job=["Captain", "Vulcan", "Fixer Guy"]

info=get_info("Kirk", name, salary, job)
for e in info:
  print(e)
```

```
35
Captain
```

Necessity of Dictionaries

- The code is messy, unpythonic
  - Easy for bug introduction
- Maintaining and passing lists to/from functions.
  - Overhead
- You have to index using integers, only.
- Updating information, may require several lists to be udpated

Dictionaries: the good life

- Instead of using lists which look like:

Index	Elements
0	E1
1	E2
2	E3
3	E4

- Use dictionaries:
  - Customized indicies

Keys	Values
Key 1	Val 1
Key 2	Val 2
Key 3	Val 3
Key 4	Val 4

Dictionaries: the good life

- Each time you add- you add data pairs
  - Storing a key, value pair

“Kirk”	“Captain”
“Spock”	“Vulcan”
“Scotty”	“Fixer Guy”

Dictionaries: the good life

```
st_dict={} #empty Dictionary
st_dict={"Kirk":"Captain", "Spock"
   :"Vulcan", "Scotty":"Fixer Guy"}
```

Dictionaries: Values

- Values can be:
  - Duplicates (many of the same value in the same dictionary)
  - Any type
    - Can even be lists, or even other dictionaries
- Keys :
  - Must be: unique!!!!
  - Immutable type-
    - Float, int, string, tuple, bool
      - Be careful using float as key
- There is no necessary order to the values/keys

```
d = {4:{1:0}, (1,3):"twelve", 'const':[3.14,2.7,8.44]}
```

Dictionaries:

```
st_dict={} #empty Dictionary
st_dict={"Kirk":"Captain", "Spock"
   :"Vulcan", "Scotty":"Fixer Guy"}

#print(st_dict["Kirko"]) THIS IS ERROR
print(st_dict["Kirk"]) # prints: Captain
```

- Create dictionary using the {}
- You can initialize a list by adding the key:value pair in between {}
- Access the values by indexing by the key

Dictionaries: Adding, Testing, Deleting

“Kirk”	“Captain”
“Spock”	“Vulcan”
“Scotty”	“Fixer Guy”

- Add an entry:
  - st_dict[“Chekov”]= “Navigator”
- Test if key in dictionary
  - “Sulu” in st_dict
    - Returns false
  - “Scotty” in st_dict
    - Returns true
- Delete an entry
  - Del(st_dict[“Scotty”])

Dictionary Operations

- Iteration:

```
st_dict={} #empty Dictionary
st_dict={"Kirk":"Captain", "Spock"
  :"Vulcan", "Scotty":"Fixer Guy"}


print(st_dict.keys())
```

```
Captain
dict_keys(['Kirk', 'Spock', 'Scotty'])
```

Dictionary Operations

```
st_dict={} #empty Dictionary
st_dict={"Kirk":"Captain",
  "Spock":"Vulcan", "Scotty"
  :"Fixer Guy"}


print(st_dict.values())
```

```
13:05:11)
[GCC 4.8.2] on linux
>
dict_values(['Captain',
'Vulcan', 'Fixer Guy'])
>
```

List vs Dictionary:

- List:
  - Ordered sequence of elements
  - Look up elements by an integer index
  - Indices have an order
  - Index is an integer
- Dictionary:
  - Matches keys to values
  - Look up one item by another item
  - No order I guaranteed
  - Key can be any IMMUTABLE type