Python 3 Modules, File I/O, Dictionaries, and other bits

Eddie Guo

September 2019

1 Introduction

(i) Map fn

(ii) Import modules

(iii) Reading/writing to files

(iv) Dictionaries

1.1 `map()`

- Unpacks elements of map obj & assigns fn to indiv vars
- Ex: split string into list then apply map fn on list

2 Modules

2.1 Introduction to Modules

- Module: file that contains Python fn defs
  - Another Python program will import module & call fns
  - Allows fns to be reused
  - Organization: related fns grouped in one file
- Modules can also contain objs that can be accessed from other files (generally constants)
- Rules for module names:
  - Filename should end in .py
  - Can't be same as Python keyword

```python
# use any fn in module
import module_name
module_name.fn1() # use dot notation when calling fn

# import specific fns in module
from module_name import fn1, fn2
fn1() # call fn w/o ref to module
```

2.2 `__name__ == "__main__"`

- Can include additional code outside fn defs in module file
  - Use `__name__ == "__main__"`
  - Only code in main program will run
- When module file run, it's main program
- Imported modules are NOT in main program

```python
# program 1
def area(width, length):
    return width*length

if __name__ == "__main__":
    print(area(3,5))
15 # output 1

# program 2
```

```python
import rectangle

if __name__ == "__main__":
    print(rectangle.area(70,100))
7000 # output 2; notice how it doesn't print out output 1
```

2.3 Standard Library Modules

Useful modules:

- time
- sys
- os
- math
- random
- pickle

2.3.1 `math` Module

- Rounding fns
  - math.ceil(x), math.floor(x), math.trunc(x)
- Trigonometric fns
  - math.cos(x), math.sin(x), math.tan(x)
  - math.acos(x), math.asin(x), math.atan(x)
  - math.degrees(x), math.radians(x)
- Constants
  - math.pi, math.e

3 File I/O

3.1 Files for Input/Output

- So far, got user input via `input()`
- Some problems req lots of data, or same data to be reused
  - Manually entering data can be tedious
  - Instead, save data to file
  - Allows program to retain data btw executions
- In gen, 2 types of files:
  - Binary
  - Text (human readable)

3.2 Using Files

1. Open connection to file → create file obj
2. Read data from file or write data to file
3. Close connection to file (else, data may not be saved)

3.2.1 Open File

- Read only (default) → "r"
- Read & write → "r+"
- Write only → "w"
- Append to end of file → "a"
- Append a "b" to above modules for binary file (ex: "rb", "t" for text (default)

```python
student_data = open('studentData.txt', 'r') # can also specifiy path in place of '
studentData.txt'
```

3.2.2 `os.path`: Check if File Exists

- B4 trying to open file, may wanna check if files exists
- Use `os.path` module: `os.path.isfile(fname)` returns True if `fname` exists

```python
import os.path

fname = input('Enter a filename: ')
while not os.path.isfile(fname):
    print('File does not exist')
    fname = input('Enter a filename: ')

fin = open(fname, 'r')
```

3.2.3 Methods to Read from File

1. `file_object_name.read(size)`
   - Reads contents of file up to `size` chars (text file)
   - If `size` not specified, will read to end of file
   - Contents returned as single string, including any `\n`
2. `file_object_name.readline()`
   - Reads single line
   - Line returned as string, including `\n` if present
3. `file_object_name.readlines()`
   - Reads all lines in file
   - Lines returned as list of strings, including `\n` if present

```python
# Example 1: views file as list
infile = open('names.txt', 'r')
for line in infile:
    line = line.strip('\n')
    print(line)
infile.close()

# Example 2: reads file into list
infile = open('names.txt', 'r')
alist = infile.read().splitlines() # splitlines() splits lines at line boundaries
# splitlines(True) includes line breaks in resulting list
for line in alist:
    print(line)
infile.close

# Both examples produce identical output
```

3.2.4 Writing to File: `file_object_namewrite(string)`

- Used to write data to file, or append data to file, depending on mode file was opened in
- Argument must be single string → use `str()` to convert (ex: if you're trying to input int, must use `file.write(str(your_int))`

3.2.5 Close File

- Always close any file you open!
  - Write: closing file flushes buffer
  - Read: Hogs resources if you don't close
- Either use `close()` or **context manager** to automatically close file when finished using (more Pythonic)

```python
# Ex: context manager -> USE THIS INSTEAD OF close()
with open('studentData.txt', 'r') as fin:
    my_data = fin.read()
```

4 Dictionaries

4.1 Built-In Type: Dictionary

- Dictionaries are collections of associated pairs of items
- Dictionaries are mutable
- Elements in dictionaries do NOT have order
- Pair consists of key & value {key: value}
- Keys must be unique & immutable
- Value can be non-unique & mutable or immutable

```python
cities = {
    'AB': ['Edmonton', 'Calgary'],
    'BC': ['Victoria', 'Vancouver', 'Richmond'],
    'ON': 'Toronto'
}
```

- Values accessed via keys: `cities['AB']`
- New pairs can be added
  - `cities['QC']='Montreal'`
- Existing values can be changed:
  - `cities['QC']='Quebec'`
- Existing pairs can be deleted:
  - `del cities['QC']`

- `list(dict_name)` returns list of keys of dictionary
- `dict_name.keys()` returns iterable keys of dictionary
- `dict_name.values()` returns iterable values of dictionary
- `dict_name.items()` returns iterable pairs (key, value) of dictionary
- `in` returns `True` or `False` depending on whether key exists