COMP10001 Foundations of Computing
Modules, List Comprehensions, and File IO

Semester 1, 2019
Tim Baldwin, Nic Geard, Farah Khan, and Marion Zalk

THE UNIVERSITY OF MELBOURNE

— version: 1537, date: April 18, 2019 —

Reminders

- Tomorrow is a public holiday (Good Friday)
- Next week is the mid-semester break
- No Grok Live(ish) Help after today, until semester resumes on Mon 29/4
- Project 1 due today
- Grok worksheets 12–13 due Monday 29/4
- Project 2 out today; due Thursday 9/5

Lecture Agenda

- Last lecture:
  - Returning early
  - Parameters and arguments
  - The call stack
- This lecture:
  - Modules
  - List comprehensions
  - File IO

Lecture Outline

1. Modules
2. Making Sense of Comprehensions
3. Files

Modules

- Modules are pre-prepared "stores" of convenient methods/variables which expand the functionality of Python
- They can take the form of local files or "standard libraries"
- To access the contents of a module, import it

```
import math

area = math.pi * radius**2

phi = (1 + math.sqrt(5))/2
```

- `import` adds the module to the local namespace

Modules

```
1 def area(radius):
2     import math
3     return math.pi * radius**2
4
5 print(area(2))
6 print(pi)
```

- What is in the local namespace of `area`?
- What is in the global namespace?
- Does line 6 work?

Modules

- It is also possible (but generally avoided) to import all methods and constants from a library into the local namespace:

```
from math import *
area = pi * radius**2
```

- Alternatively you can selectively import objects:

```
from math import pi, e
area = pi * radius**2
```

including the possibility of renaming them:

```
from math import pi as mypi
area = mypi * radius**2
```


Painless Dictionaries: `defaultdict`

- Alternatively, simplify your life with `defaultdict`:

```
from collections import defaultdict

def count_digits(num):
    '''Count the digits in number `num`'''
    digit_count = defaultdict(int)
    for digit in str(num):
        digit_count[digit] += 1
    return digit_count
```


List Comprehensions

- We are often constructing lists in Python using `for` loops, e.g.:

```
mylist = []
for key, val in mydict.items():
    mylist.append(val, key)
```

- A **list comprehension** allows us to do this in a single line:

```
[val, key for key, val in mydict.items()]
```

- We can also nest for loops:

```
[v+s for s in 'SHDC' for v in 'A234567890JQK']
```

or filter elements:

```
[i for i in range(-9, 10, 2) if not (i % 3)]
```


Painless Dictionaries: `defaultdict`

- When using dictionaries as "accumulators" you need to initialise every value for new keys ...

```
def count_digits(num):
    '''Count the digits in number `num`'''
    digit_count = {}
    for digit in str(num):
        if digit in digit_count:
            digit_count[digit] += 1
        else:
            digit_count[digit] = 1
    return digit_count
```


Lecture Outline

1 Modules

2 Making Sense of Comprehensions

3 Files

Other Comprehension Types

- We use the same basic syntax for **dictionary comprehensions**:

```
{value: key for key, value in d.items()}
```

... and **set comprehensions**:

```
{abs(i) for i in range(-5, 5, 2)}
```

Lecture Outline

1. Modules
2. Making Sense of Comprehensions
3. Files

Files

- Computer memory is volatile
  - RAM, cache, registers
  - Power off, it is all gone
  - Python program finishes, it is all gone
- Computers use "disk" for longer term storage
  - A physical hard disk that mechanically spins
  - Flash drives without mechanical parts
  - "The cloud" (sends it off to a physical disk)
- Disks have "filesystems" (except on systems such as iOS, where data is linked to an app)
- Python can read and write files

Reading Files

- Reading data in from a (local) file:

```
FILENAME = 'jabberwocky.txt'
text = open(FILENAME).read()
lines = open(FILENAME).readlines()
fp = open(FILENAME)

for line in fp:
    ...
```

- `read()` reads the entire file
- `readlines()` reads into a list of lines
- `for` iterates over lines in the file

File Writing/Appending

- Writing to a file:

```
FILENAME = 'file.txt'
text = open(FILENAME,'w')
text.write('Tim woz ere')
text.close()
```

- Appending to a file:

```
FILENAME = 'file.txt'
text = open(FILENAME,'a')
text.write('Tim woz ere again')
text.close()
```


File Generators

- Even better, you can use a "generator":

```
with open('file.txt') as f:
    for line in f:
        print(line)
```

with the advantage that:

- it automatically closes the file
- it is more efficient than direct iteration

Persistent Storage

- Sometimes, we may want to store data in a native Python format to be able to access directly again in the future/share data
- One way of doing this is via JSON, which can be used to encode arbitrarily nested Python lists/tuples/dictionaries, e.g. one record at a time:

```
import json

with open("data.json", "w") as out:
    data1 = [(2, 1), {"a": 3.0}]
    json.dump(data, out)
```

Persistent Storage

- It can then be accessed by:

```
import json

with open("data.json") as f:
    data = json.load(f)
```


Lecture Summary

- What are modules?
- What are standard libraries?
- What are list comprehensions, and how do they relate to standard list construction methods?
- How do you read/write to a file?