COMP10001 Foundations of Computing
Modules and Files

Semester 2, 2018

Chris Leckie & Nic Geard

Reminders

- Workshops 7 and 8 due 23:59 Monday 27/8.
- Solutions for Practice Project available soon
- Project 1 open in Grok
- Advice: start the project NOW
- You must complete the "Academic Integrity videos and quizzes" in Grok before the deadline for Project 1

Lecture Agenda

- Last lecture:
  - Advanced Functions
  - Namespaces
- This lecture:
  - Modules
  - File access
  - Python tips/tricks

Modules I

- Modules are pre-prepared "stores" of convenient methods/variables which expand the functionality of Python
- 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 II

```
def area(radius):
    import math
    return(math.pi * radius**2)

print(area(2))
print(pi)
```

- What is in the local namespace of `area`?
- What is in the global namespace?
- Does the 5th line work?

Modules III

- 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
```

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 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) \\ a "file pointer"

for line in fp:
    ...
```

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

Challenge: given a file of tramstops and routes, print a list of routes for each stop.

- Each line of the file has two numbers, the stop and then the route.
- Hint: `split()` method of a string breaks the string into a list of words.

(At least 10 minutes work here, so focus!)

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()
```

- More after the break...

That accumulating loop again

- Did you notice that the loop we used just now had that same structure as last time?

```
Initialise an accumulator variable
for x in iterable:
    if some condition on x:
        Alter the accumulator somehow
Now do something with the accumulator
```

- This is a very common code pattern
- Accumulators can be integers, floats, lists, dictionaries, ...

defaultdict I

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

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

defaultdict II

- Without `defaultdict`

```
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)
```

Python Tips: List Comprehensions

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

```
mylist = []
for i in range(-4,6,2):
    mylist.append(i)
```

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

```
[i for i in range(-4,6,2)]
```

- It also allows us to filter elements in a list:

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

Python Tips: Assignment

- We often apply some operation to a variable and assign the result back to that variable:

```
i = i - 1
j = j * 2
mystr = mystr + letter
```

There is a family of convenient "shorthands" for this, one for each binary operator:

```
i -= 1
j *= 2
mystr += letter
```

Lecture Summary

- What are modules?
- How do you read a file?
- What is `defaultdict` and why is it useful?
- What are list comprehensions and why are they useful?
- What does `+=` and `*=` mean?