Files

CSC 1200 - Principles of Computing

Overview

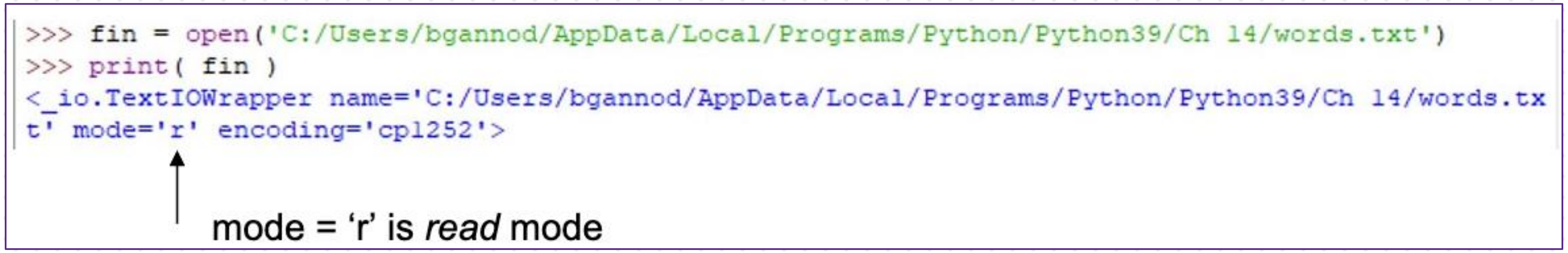
- Persistence
- Reading From a File with *readline* and *read*
- Writing Files
- The *with* and *for* Statements
- Format Operator

Persistence

- Most of the programs we have seen so far are transient – they run and produce output, but when they end the data disappears.
- Other programs are **persistent** – they run for a long time (or all the time) and they keep at least some of their data in permanent memory (a hard drive, for example)
  - Operating systems
  - Web servers
- One of the simplest ways for programs to maintain their data is by reading and writing text files. (An alternative is to use a database, which is a topic for another course.)

Reading From a File with *readline*

- The built-in function open takes the name of a file as a parameter and returns a file object that you can use to read the file.

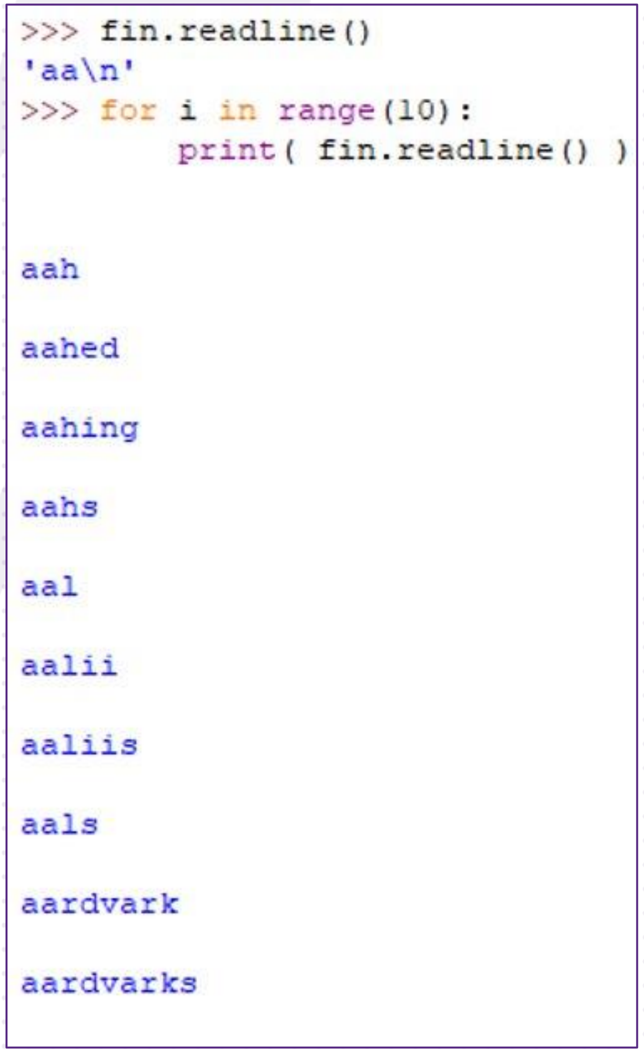


```
>>> fin = open('C:/Users/bgannod/AppData/Local/Programs/Python/Python39/Ch 14/words.txt')
>>> print( fin )
<_io.TextIOWrapper name='C:/Users/bgannod/AppData/Local/Programs/Python/Python39/Ch 14/words.tx
t' mode='r' encoding='cp1252'>
```

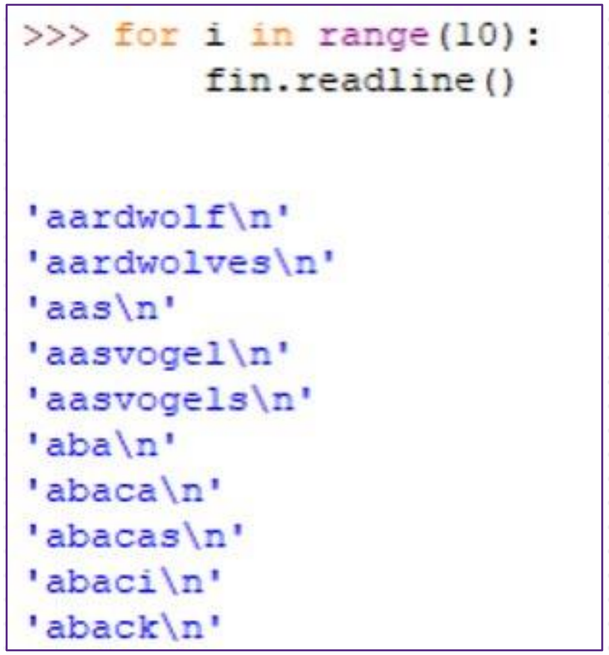
mode = ‘r’ is *read* mode

- The method readline is used to read one line of the file.
  - A newline character ‘\n’ indicates the end of the line.
  - The newline character is read in as part of the line.
  - If you want to remove the newline, you can use the strip method for str objects.
- Use the close method to close the file.

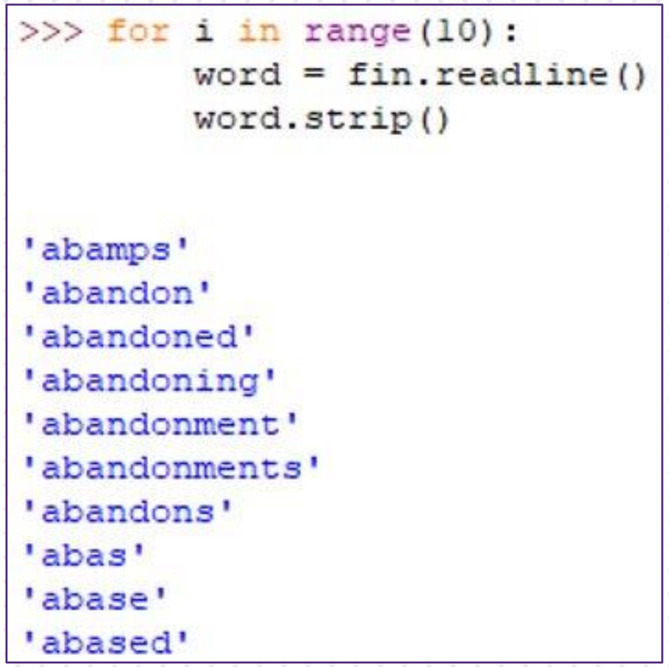
Example



```
>>> fin.readline()
'aa\n'
>>> for i in range(10):
        print( fin.readline() )


aah

aahed

aahing

aahs

aal

aalii

aaliis

aals

aardvark

aardvarks
```



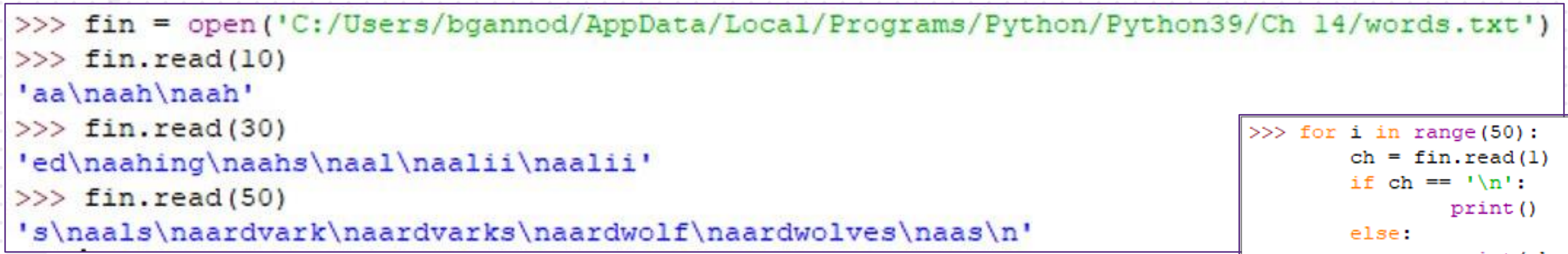
```
>>> for i in range(10):
        fin.readline()


'aardwolf\n'
'aardwolves\n'
'aas\n'
'aasvogel\n'
'aasvogels\n'
'aba\n'
'abaca\n'
'abacas\n'
'abaci\n'
'aback\n'
```



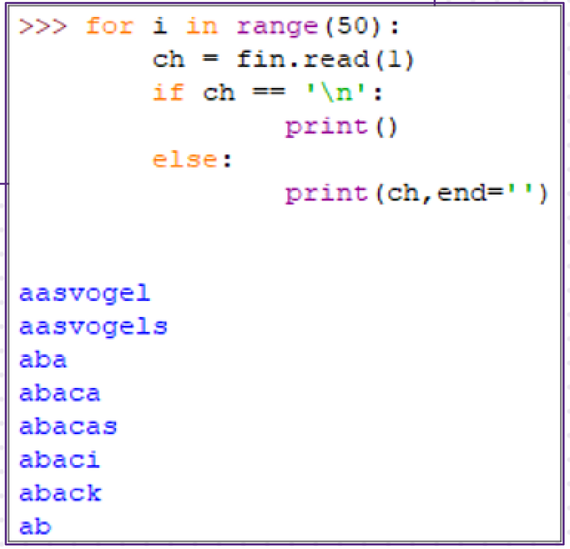
```
>>> for i in range(10):
        word = fin.readline()
        word.strip()


'abamps'
'abandon'
'abandoned'
'abandoning'
'abandonment'
'abandonments'
'abandons'
'abas'
'abase'
'abased'
```

Reading From a File Using *read*

- The *read* method can be used to read a specified number of characters.
- If the number of characters is not specified, the entire file will be read in (if possible).



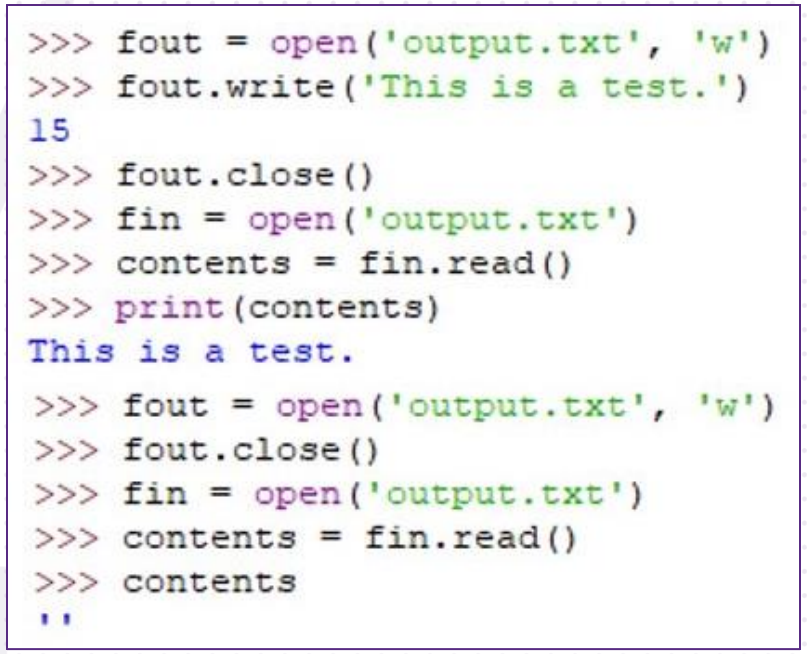
```
>>> fin = open('C:/Users/bgannod/AppData/Local/Programs/Python/Python39/Ch 14/words.txt')
>>> fin.read(10)
'aa\naah\naah'
>>> fin.read(30)
'ed\naahing\naahs\naal\naalii\naalii'
>>> fin.read(50)
's\naals\naardvark\naardvarks\naardwolf\naardwolves\naas\n'
```



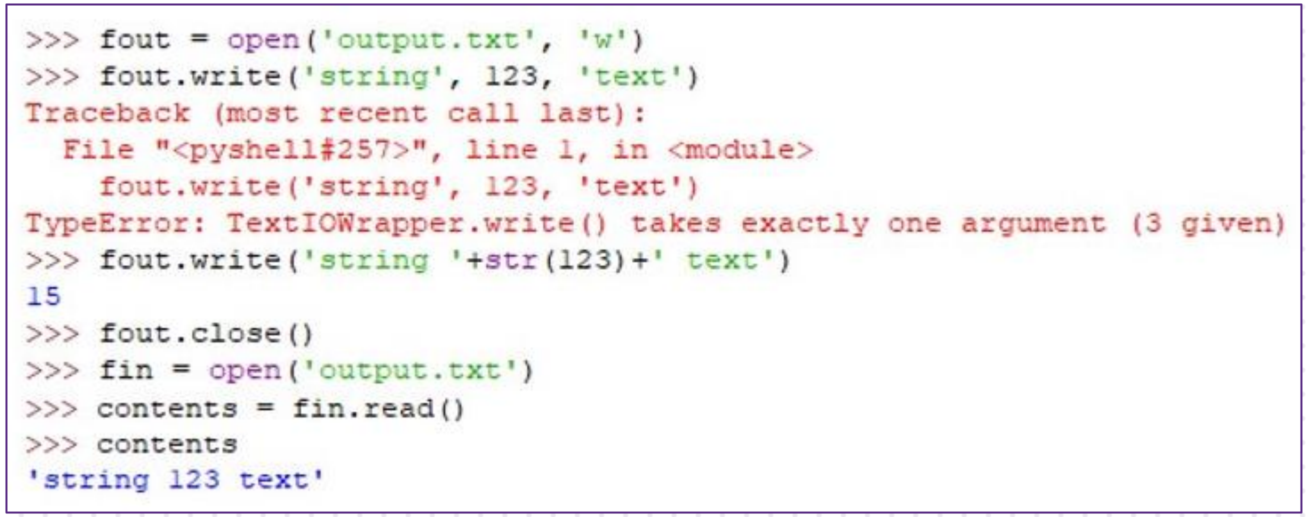
```
>>> for i in range(50):
	ch = fin.read(1)
	if ch == '\n':
		print()
	else:
		print(ch,end='')


aasvogel
aasvogels
aba
abaca
abacas
abaci
aback
ab
```

Writing Files

- The default mode for open is read mode. If you want to write a file, you have to open it with mode ‘w’ as a second parameter.
  - If the file already exists, opening it in write mode clears out the old data and starts fresh.
  - If the file does not exist, a new one is created.
- You use the write method to write **a string** into the file.



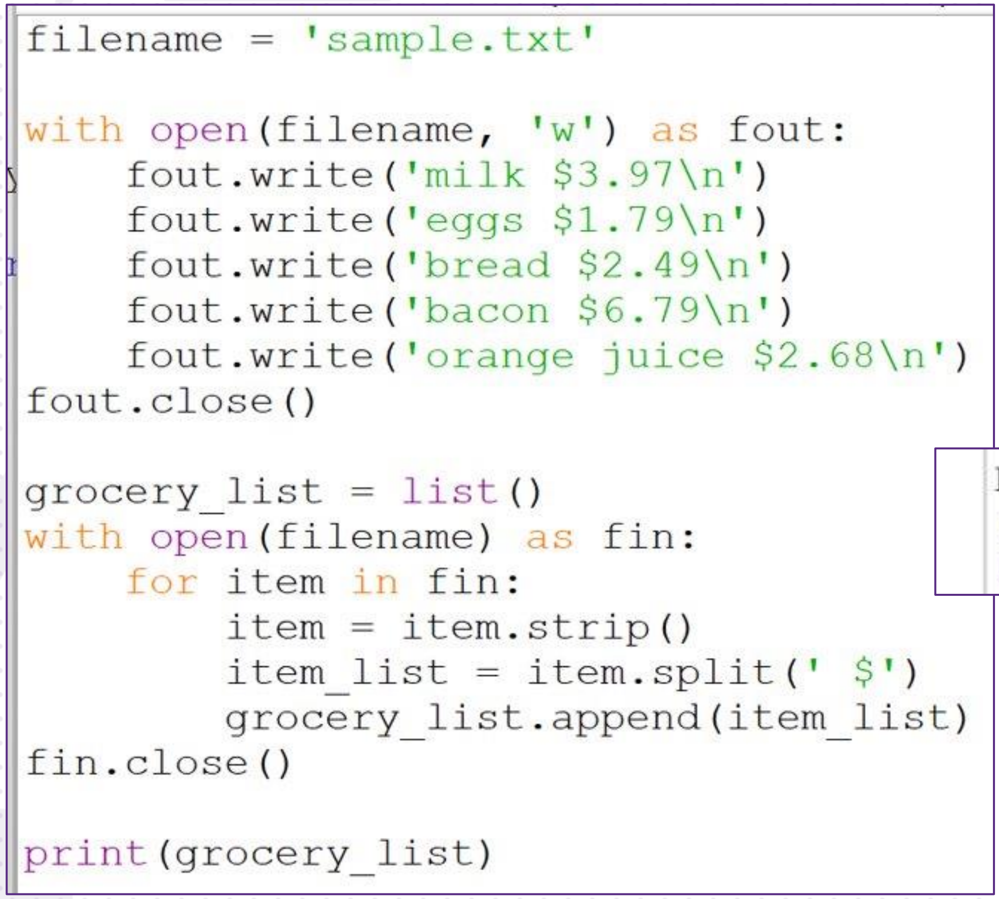
```
>>> fout = open('output.txt', 'w')
>>> fout.write('This is a test.')
15
>>> fout.close()
>>> fin = open('output.txt')
>>> contents = fin.read()
>>> print(contents)
This is a test.
>>> fout = open('output.txt', 'w')
>>> fout.close()
>>> fin = open('output.txt')
>>> contents = fin.read()
>>> contents
''
```



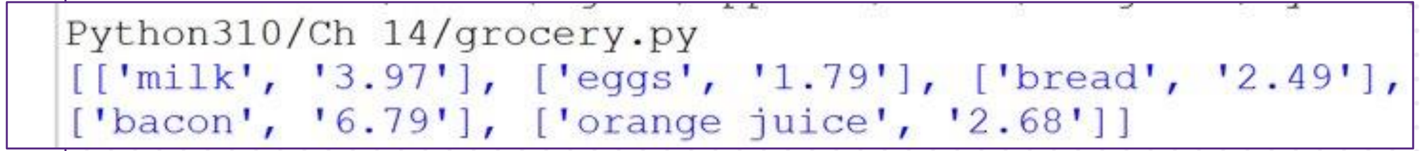
```
>>> fout = open('output.txt', 'w')
>>> fout.write('string', 123, 'text')
Traceback (most recent call last):
  File "<pyshell#257>", line 1, in <module>
    fout.write('string', 123, 'text')
TypeError: TextIOWrapper.write() takes exactly one argument (3 given)
>>> fout.write('string '+str(123)+' text')
15
>>> fout.close()
>>> fin = open('output.txt')
>>> contents = fin.read()
>>> contents
'string 123 text'
```

The *with* and *for* Statements

You can use the with statement to open read/write a file and for to traverse.



```
filename = 'sample.txt'

with open(filename, 'w') as fout:
    fout.write('milk $3.97\n')
    fout.write('eggs $1.79\n')
    fout.write('bread $2.49\n')
    fout.write('bacon $6.79\n')
    fout.write('orange juice $2.68\n')
fout.close()

grocery_list = list()
with open(filename) as fin:
    for item in fin:
        item = item.strip()
        item_list = item.split(' $')
        grocery_list.append(item_list)
fin.close()

print(grocery_list)
```

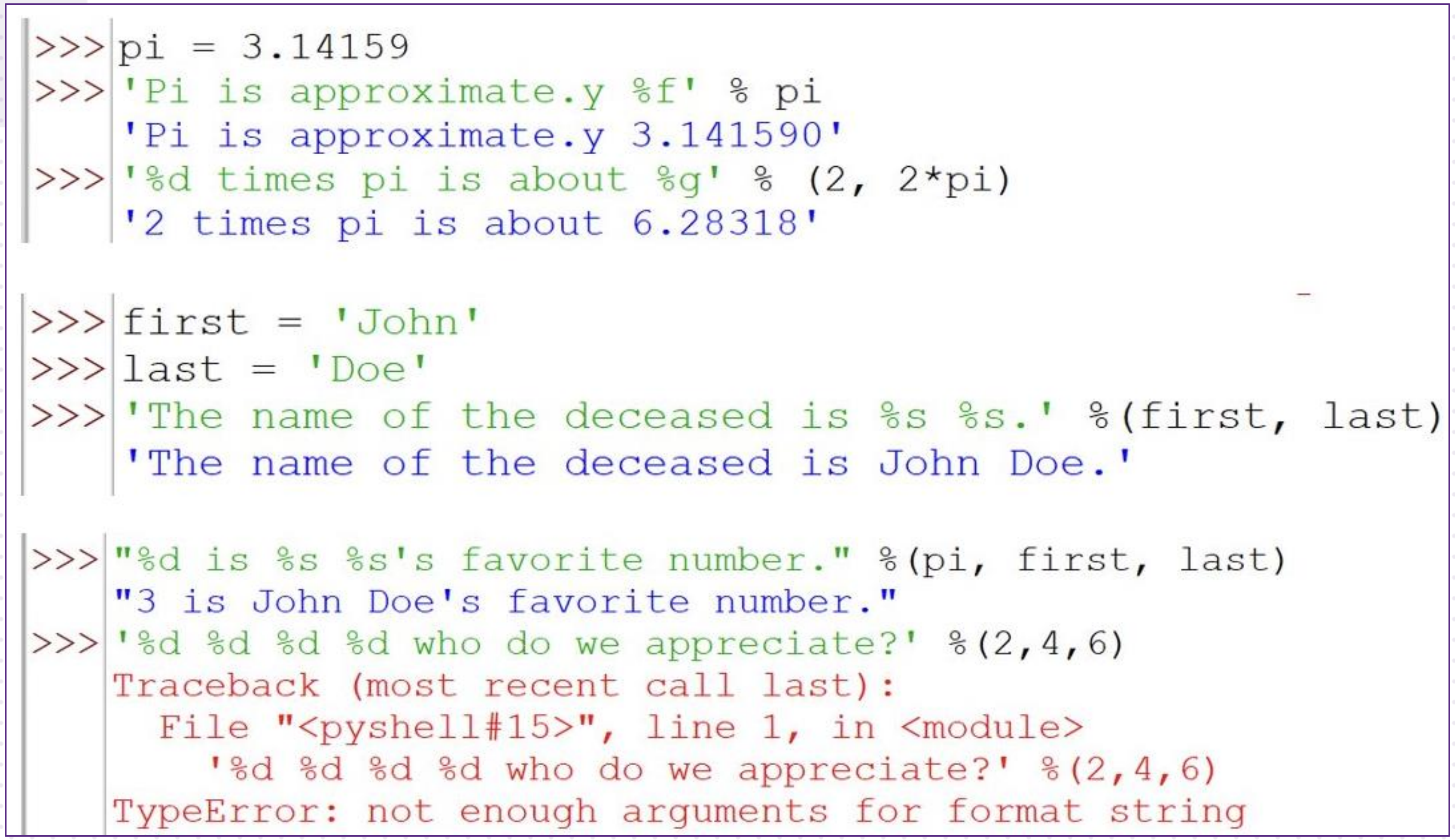


```
Python310/Ch 14/grocery.py
[['milk', '3.97'], ['eggs', '1.79'], ['bread', '2.49'],
['bacon', '6.79'], ['orange juice', '2.68']]
```

Format Operator

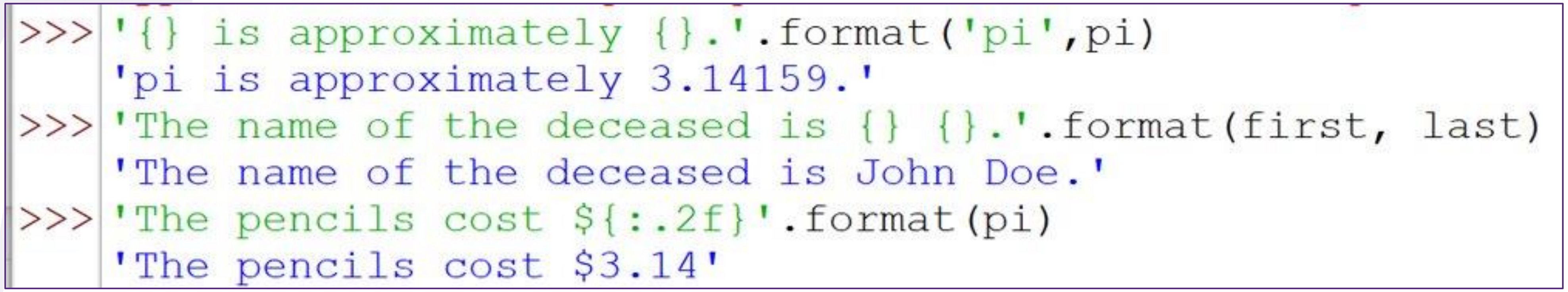
- The argument of write has to be a string. An alternative to converting all values to strings and concatenating them is to use the **format operator**.
- The format operator is denoted by % (% is the modulus operator when applied to integers, but the format operator when applied to strings.)
    - The first operand is the **format string**, which contains one or more **format sequences**.
    - Format sequences:
        - %d is used for integers
        - %f or %g are used for floating point numbers
        - %s is used for strings
- Formatting can also be done using .format instead of % (this is probably less confusing!)

Examples Using %

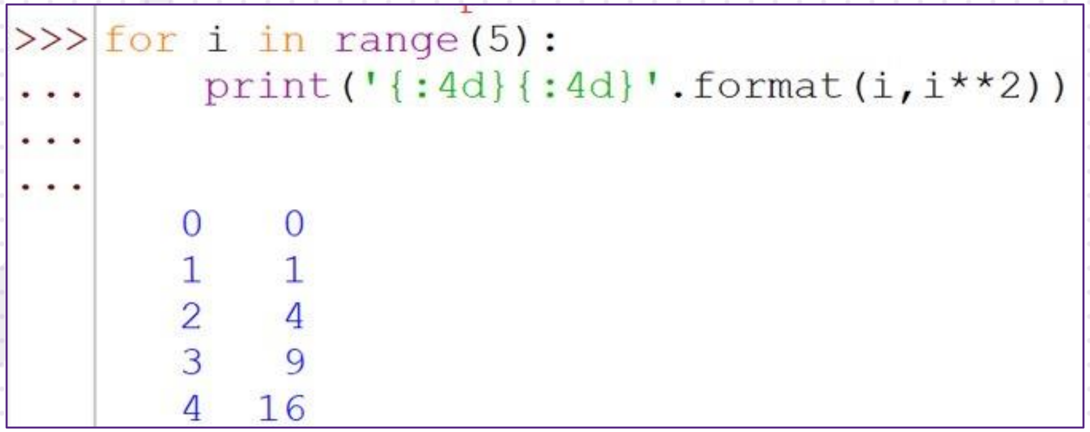


```
>>> pi = 3.14159
>>> 'Pi is approximate.y %f' % pi
    'Pi is approximate.y 3.141590'
>>> '%d times pi is about %g' % (2, 2*pi)
    '2 times pi is about 6.28318'

>>> first = 'John'
>>> last = 'Doe'
>>> 'The name of the deceased is %s %s.' %(first, last)
    'The name of the deceased is John Doe.'

>>> "%d is %s %s's favorite number." %(pi, first, last)
    "3 is John Doe's favorite number."
>>> '%d %d %d %d who do we appreciate?' %(2,4,6)
    Traceback (most recent call last):
      File "<pyshell#15>", line 1, in <module>
        '%d %d %d %d who do we appreciate?' %(2,4,6)
    TypeError: not enough arguments for format string
```

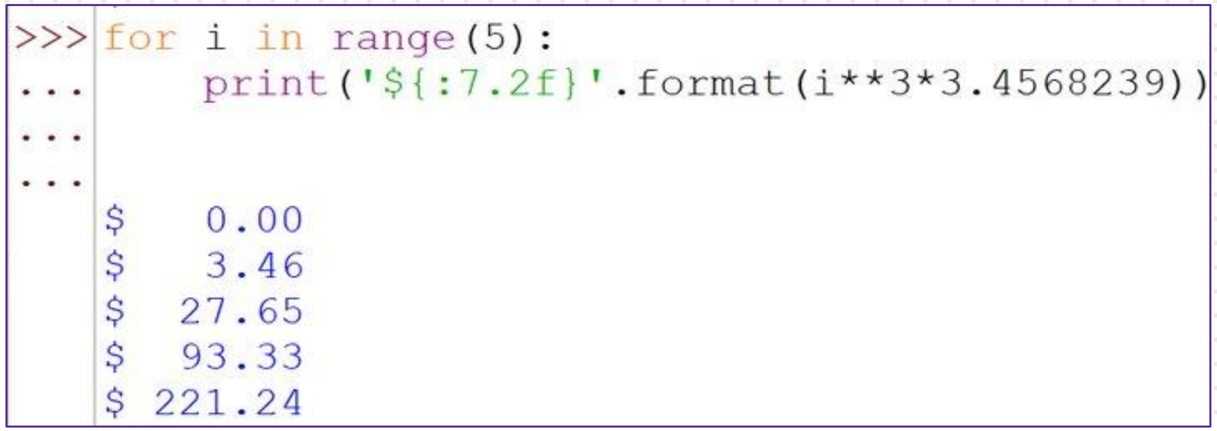
Examples Using *.format*



```
>>> '{} is approximately {}.'.format('pi',pi)
'pi is approximately 3.14159.'
>>> 'The name of the deceased is {} {}.'.format(first, last)
'The name of the deceased is John Doe.'
>>> 'The pencils cost ${:.2f}'.format(pi)
'The pencils cost $3.14'
```



```
>>> for i in range(5):
...     print('{:4d}{:4d}'.format(i,i**2))
...
...
   0   0
   1   1
   2   4
   3   9
   4  16
```



```
>>> for i in range(5):
...     print('${:7.2f}'.format(i**3*3.4568239))
...
...
$   0.00
$   3.46
$  27.65
$  93.33
$ 221.24
```

Other Modes

- Mode ‘r’ - opens a file in read only mode (this is the default mode)
- Mode ‘r+’ - opens a file in read/write mode; starts at the beginning of the file; raises an error if the file does not exist.
- Mode ‘w’ - opens a file in write only mode.
- Mode ‘w+’ - opens a file in write/read mode; if the file already exists, the data is overwritten; if file does not exist a new file is created
- Mode ‘a’ - open a file for writing in append mode; start writing at the end of the file (after existing data)
- Mode ‘a+’ - open a file for reading and writing in append mode.