Lecture 6: File Processing and Exception Handling

CS5001 / CS5003:
Intensive Foundations
of Computer Science

```
>>> try:
...     with open("not-a-file-name") as f:
...         text = f.read()
... except FileNotFoundError:
...     print("Could not open file!")
...
Could not open file!
>>>
```

```
>>> with open("/usr/share/dict/words") as f:
...     words = f.readlines()
...
>>> words = [x[:-1] for x in words]
>>> words[0]
'A'
>>> words[1]
'a'
>>> words[-1]
'Zyzzogeton'
>>>
```

PDF of this presentation

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- let's first go over some of the solutions to the lab.

Starter:

```
def square():
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- let's first go over some of the solutions to the lab.

Starter:

```
def square():
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
```

Better:

```
def square(side_length, initial_rotation=0):
    turtle.left(initial_rotation)
    for i in range(4):
        turtle.forward(side_length)
        turtle.left(90)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- let's first go over some of the solutions to the lab.

Starter:

```
def triangle(side_length):
    pass


def pentagon(side_length):
    pass


def polygon(side_length, num_sides):
    pass
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- let's first go over some of the solutions to the lab.

Starter:

```
def triangle(side_length):
    pass


def pentagon(side_length):
    pass


def polygon(side_length, num_sides):
    pass
```

Solution:

```
def triangle(side_length):
    for i in range(3):
        turtle.forward(side_length)
        turtle.left(120)


def pentagon(side_length):
    for i in range(5):
        turtle.forward(side_length)
        turtle.left(72)


def polygon(side_length, num_sides):
    angle = 360 / num_sides
    for i in range(num_sides):
        turtle.forward(side_length)
        turtle.left(angle)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- let's first go over some of the solutions to the lab.

Starter:

```
def circle(radius):
    NUM_SIDES = 100
    pass
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- let's first go over some of the solutions to the lab.

Starter:

```
def circle(radius):
    NUM_SIDES = 100
    pass
```

Solution:

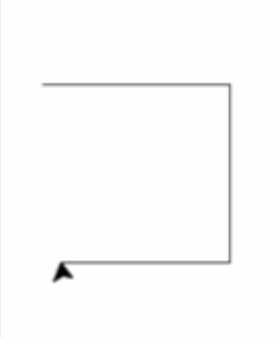
```
def circle(radius):
    NUM_SIDES = 100
    circumference = 2 * radius * math.pi
    side_length = circumference / NUM_SIDES
    polygon(side_length, NUM_SIDES)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- let's first go over some of the solutions to the lab.

Starter:

```
def draw_spiral(line_len):
    pass
```

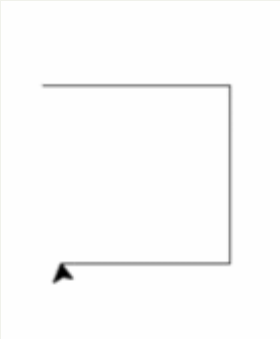


Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- let's first go over some of the solutions to the lab.

Starter:

```
def draw_spiral(line_len):
    pass
```



Solution:

```
def draw_spiral(line_len):
    if line_len > 0:
        turtle.forward(line_len)
        turtle.right(90)
        draw_spiral(line_len - 5)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def circle(radius):
    NUM_SIDES = 100
    circum = 2 * radius * 3.14
    side_length = circum / NUM_SIDES
    polygon(side_length, NUM_SIDES)
```

How could we make this better?

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def circle(radius):
    NUM_SIDES = 100
    circum = 2 * radius * 3.14
    side_length = circum / NUM_SIDES
    polygon(side_length, NUM_SIDES)
```

How could we make this better?

Use the defined constant, **`math.pi`**

```
def circle(radius):
    NUM_SIDES = 100
    circum = 2 * radius * math.pi
    side_length = circum / NUM_SIDES
    polygon(side_length, NUM_SIDES)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def circle (radius):
    NUM_SIDES = 100
    pi = math.pi
    side_length = 2*pi*radius /NUM_SIDES
    polygon(side_length, NUM_SIDES)
```

How could we make this better?

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def circle (radius):
    NUM_SIDES = 100
    pi = math.pi
    side_length = 2*pi*radius /NUM_SIDES
    polygon(side_length, NUM_SIDES)
```

How could we make this better?

1. Spaces between operator and operands
2. Can just use **`math.pi`** without redefining.

```
def circle (radius):
    NUM_SIDES = 100
    side_length = 2 * math.pi * radius / NUM_SIDES
    polygon(side_length, NUM_SIDES)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def draw_spiral(side_length):
    if side_length > 0:
        for x in range(3):
            turtle.forward(side_length)
            turtle.left(90)
            draw_spiral(side_length - 5)
    else:
        quit()
```

How could we make this better?

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def draw_spiral(side_length):
    if side_length > 0:
        for x in range(3):
            turtle.forward(side_length)
            turtle.left(90)
            draw_spiral(side_length - 5)
    else:
        quit()
```

How could we make this better?

1. No need for loop
2. No need for **`else`** or **`quit()`**

```
def draw_spiral(side_length):
    if side_length > 0:
        turtle.forward(side_length)
        turtle.left(90)
        draw_spiral(side_length - 5)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def pentagon(side_length, initial_rotation=0):
    for i in range(5):
        turtle.left(initial_rotation)
        turtle.forward(side_length)
        turtle.right(72 + initial_rotation)
```

How could we make this better?

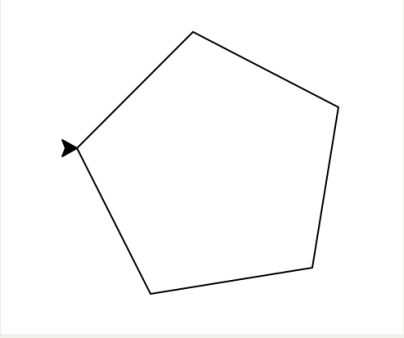
Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def pentagon(side_length, initial_rotation=0):
    for i in range(5):
        turtle.left(initial_rotation)
        turtle.forward(side_length)
        turtle.right(72 + initial_rotation)
```

How could we make this better?

```
pentagon(100, 45)
```



1. Can rotate initially outside the loop:

```
def pentagon(side_length, initial_rotation=0):
    turtle.left(initial_rotation)
    for i in range(5):
        turtle.forward(side_length)
        turtle.right(72)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def polygon(side_length, num_sides):
    for i in range(num_sides):
        turtle.forward(side_length)
        turtle.left(180 - ((num_sides - 2) * 180)/(num_sides))
```

How could we make this better?

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def polygon(side_length, num_sides):
    for i in range(num_sides):
        turtle.forward(side_length)
        turtle.left(180 - ((num_sides - 2) * 180)/(num_sides))
```

How could we make this better?

Fewer parentheses:

```
def polygon(side_length, num_sides):
    for i in range(num_sides):
        turtle.forward(side_length)
        turtle.left(180 - ((num_sides - 2) * 180) / num_sides)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def circle(radius):
    NUM_SIDES = 100
    side_length = (2 * math.pi * radius) / 100
    for i in range(NUM_SIDES):
        turtle.forward(side_length)
        turtle.left(360 / NUM_SIDES)
```

How could we make this better?

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def circle(radius):
    NUM_SIDES = 100
    side_length = (2 * math.pi * radius) / 100
    for i in range(NUM_SIDES):
        turtle.forward(side_length)
        turtle.left(360 / NUM_SIDES)
```

How could we make this better?

Use the constants in all cases

```
def circle(radius):
    NUM_SIDES = 100
    side_length = (2 * math.pi * radius) / NUM_SIDES
    for i in range(NUM_SIDES):
        turtle.forward(side_length)
        turtle.left(360 / NUM_SIDES)
```

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def draw_spiral(line_len, tweak, howmany):
    if howmany < 1:
        return square(line_len)
    turtle.right(tweak)
    draw_spiral(line_len + 1, tweak, howmany - 1)
```

How could we make this better?

Lecture 6: Lab Review

- There were a few errors and misconceptions in the lab last week -- now, let's see how some of you solved the code, and let's look at some errors and some style issues.

```
def draw_spiral(line_len, tweak, howmany):
    if howmany < 1:
        return square(line_len)
    turtle.right(tweak)
    draw_spiral(line_len + 1, tweak, howmany - 1)
```

How could we make this better?

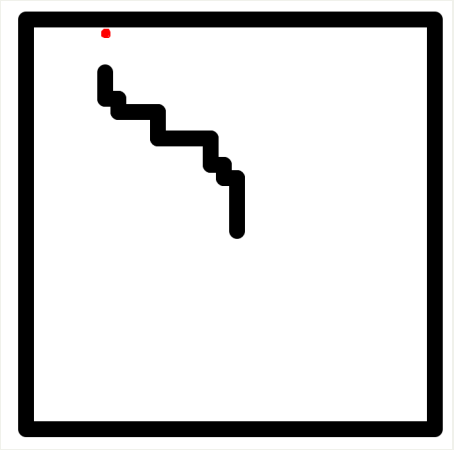
1. **`square()`** does not return a value
2. Can just use **`line_len`** to determine base case
3. No need for **`tweak`**

```
def draw_spiral(line_len):
    if line_len > 0:
        turtle.forward(line_len)
        turtle.right(90)
        draw_spiral(line_len - 5)
```

Lecture 6: Snake Homeworks 5 & 6

This week and next week's homework is to build a snake game. Fun!

The first version uses the **`Turtle`** module, which is slow. :(But, you only have to make the snake move and not go past the walls:



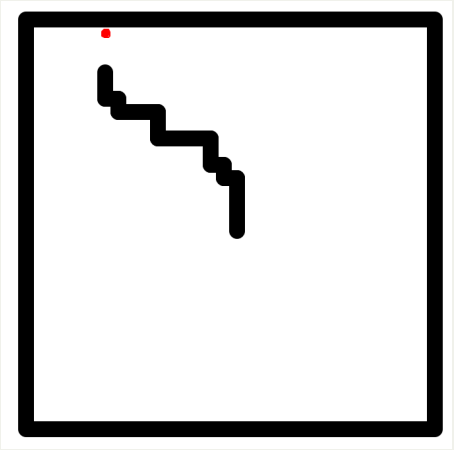
The snake board is a 2-dimensional list:

```
board = [['empty'] * settings.BOARD_WIDTH
             for i in range(settings.BOARD_HEIGHT)]
```

Lecture 6: Snake Homeworks 5 & 6

This week and next week's homework is to build a snake game. Fun!

The first version uses the **`Turtle`** module, which is slow. :(But, you only have to make the snake move and not go past the walls:



The snake board is a 2-dimensional list:

```
board = [['empty'] * 4 for i in range(3)]
```

	col 0	col 1	col 2	col 3
row 0	'empty'	'empty'	'empty'	'empty'
row 1	'empty'	'empty'	'empty'	'empty'
row 2	'empty'	'empty'	'empty'	'empty'

To access an element, you use two brackets, with the row first, then the column:

```
board[1][2] = 'food'
```

	col 0	col 1	col 2	col 3
row 0	'empty'	'empty'	'empty'	'empty'
row 1	'empty'	'empty'	'food'	'empty'
row 2	'empty'	'empty'	'empty'	'empty'

Lecture 6: Snake Homeworks 5 & 6

The snake's state (in other words, the details about the snake itself) are defined as follows, in a python dict:

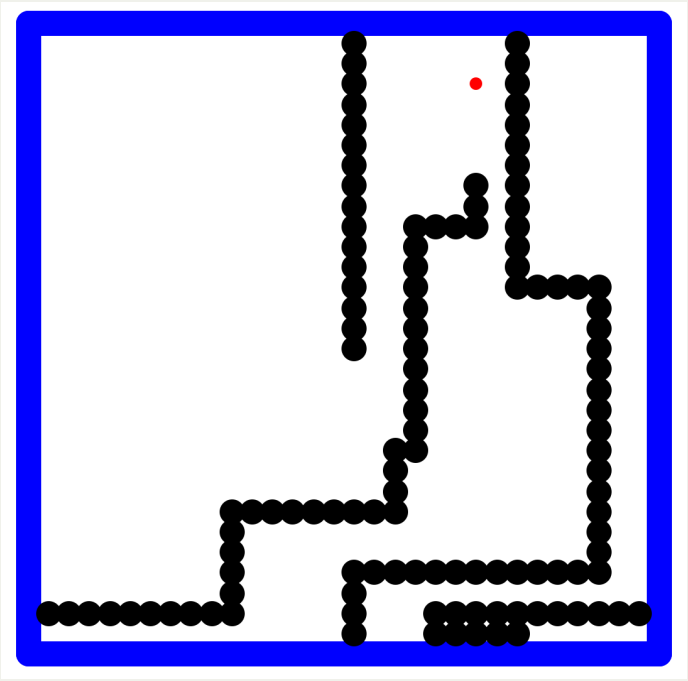
```
   snake_state = {
        'row': settings.BOARD_HEIGHT // 2,
        'col': settings.BOARD_WIDTH // 2,
        'energy': settings.START_ENERGY,
        'score': 0,
    }
```

If you want to determine what the board value is under the row and column for the snake, you could do the following:

```
under_board_value = board[snake_state['row']][snake_state['col']]
```

Lecture 6: Snake Homeworks 5 & 6

For this week's assignment, you need to *wrap* the snake around the screen:



Note the better graphics! It is also faster.

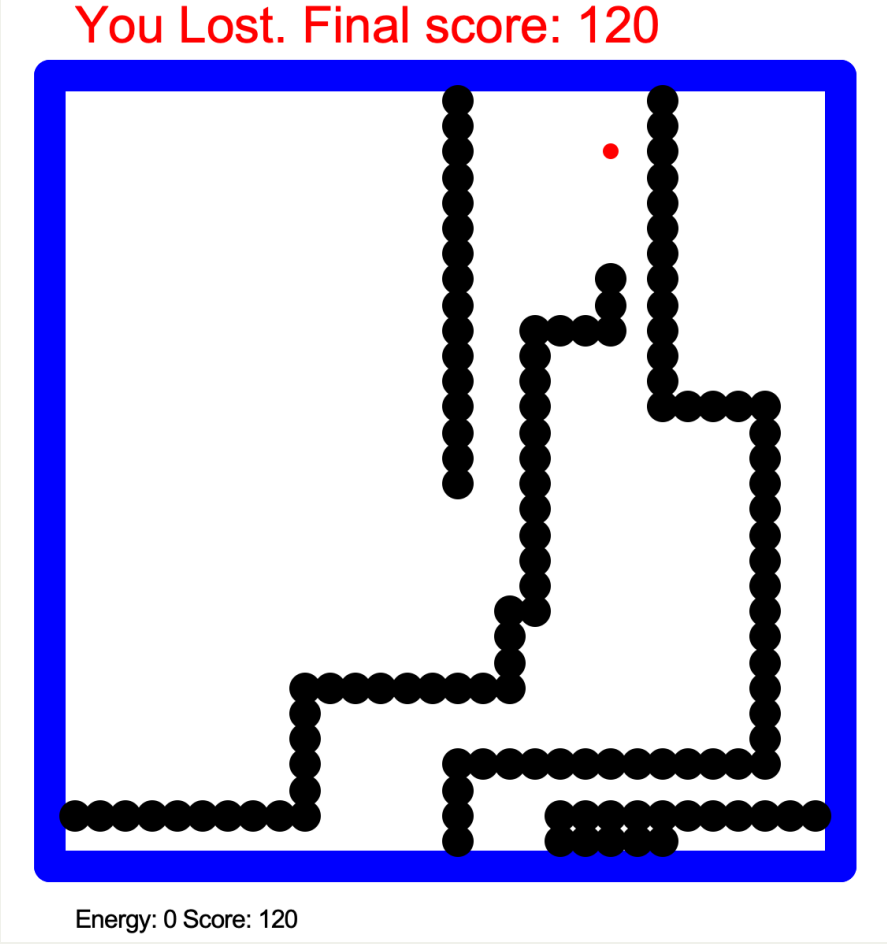
For the game to the left, the snake started in the middle and went straight up, then it wrapped around to the bottom. It also wrapped from right to left at the bottom.

How do you wrap the snake? One way is to use the remainder operator, %

Let's say the board was 30 wide, and you can move from `col == 0` to `col == 29`. What can you do when you are going right and you need to wrap? Let's say you set the value of `col` to `30`. You're off the board! But, what is `30 % 30`? `0`, which is back to the left side of the board. You've wrapped!

Lecture 6: Snake Homeworks 5 & 6

For this week's assignment, you should write all the logic for the game.



The main logic:

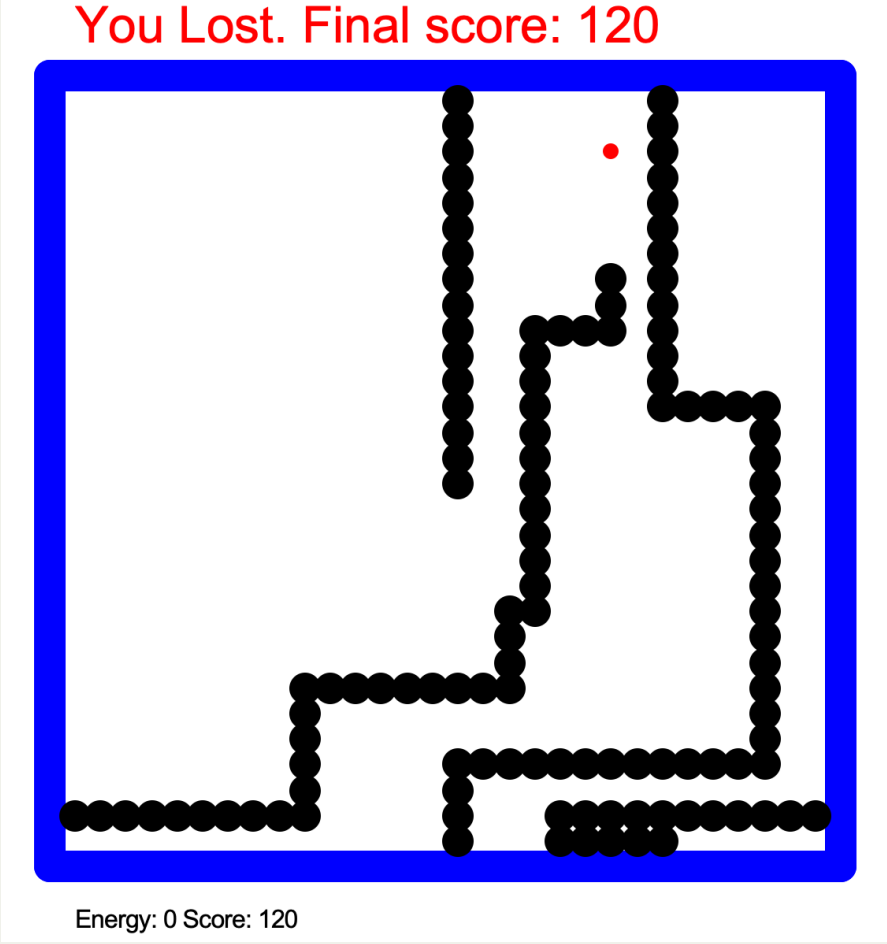
- Every time you move the snake, you lose one energy level, but you score one point.
- When you eat a red food pellet, you get 15 more energy levels.
- If you run out of energy, *or* you hit your own snake, you lose!
- It is *very* difficult to win (e.g., fill the entire board with snake)

Writing:

- Every move, you should update the energy and score
- When you die, you put a "You Lost" message.

Lecture 6: Snake Homeworks 5 & 6

Let's look at the starter code for Assignment 6



Lecture 6: File Processing

Python has the ability to read and write to *files* on your hard disk. Files are *persistent* and retain their data when your computer is off, or crashes, etc. You could not have a robust operating system without some form of persistence.

To write to a file in Python, you first have to *open* the file, and you have to also tell Python you want to write to the file. You can do this as follows:

```
>>> f = open("myfile.txt", "w"):
>>> print(f)
<_io.TextIOWrapper name='myfile.txt' mode='w' encoding='UTF-8'>
```

You have to be a bit careful with the "**w**" *flag*, because if the file already exists, it will delete it first in order to write to it. You can use the "**x**" flag if you want to ensure a file does not get deleted if you try to open it before writing.

Lecture 6: File Processing

To write to the file, you use the **`write`** function. You should then **`close`** the file after you have written to it:

```
>>> f = open("myfile.txt", "w"):
>>> f.write("This is the text I want to put into the file.\n")
46
>>> f.close()
```

The **`46`** above means that 46 characters were written to the file. The **`write`** function does not automatically put a newline on the end of strings, so you have to add it manually.

Just like with **`print`**, you can format what you write to the file:

```
>>> f = open("myfile.txt","w")
>>> name="Chris"
>>> f.write(f"My name is {name}.\n")
18
>>> f.close()
```

Lecture 6: File Processing

To read from a file, you use the **`read`** function. You first should open the file with the read flag, "**`r`**". You should also **`close`** the file after you have read from it:

```
>>> f = open("myfile.txt","r")
>>> text = f.read()
>>> f.close()
>>> print(text)
My name is Chris.
```

In the above example, we read in the entire file at once. You can also read in one line at a time with the **`readline`** function. Assume a file has the opening of *A Tale of Two Cities*:

```
>>> f = open("tale_of_two_cities.txt","r")
>>> f.readline()
'It was the best of times,\n'
>>> f.readline()
'it was the worst of times,\n'
>>> f.readline()
'it was the age of wisdom,\n'
>>> f.close()
```

Notice that the newline is *not* removed when you read an entire line.

Lecture 6: File Processing

You can also write to a file without removing its contents -- you can *append* to the end of a file, by opening it with the "**a**" flag:

```
>>> f = open("myfile.txt", "r")
>>> text = f.read()
>>> f.close()
>>> print(text)
My name is Chris

>>> f = open("myfile.txt", "a")
>>> f.write("Green Eggs and Ham\n")
19
>>> f.close()
>>> f = open("myfile.txt", "r")
>>> text = f.read()
>>> f.close()
>>> print(text)
My name is Chris
Green Eggs and Ham

>>>
```

In this case, we simply added to the end of the file, without removing the original contents.

Lecture 6: File Processing

There is actually a much more common way to read from a file in Python. It uses the same ideas, but handles some things for you (e.g., closing the file):

```
>>> with open("tale_of_two_cities.txt", "r") as f:
...     for line in f:
...         print(line)
...
It was the best of times,

it was the worst of times,

it was the age of wisdom,

it was the age of foolishness,

it was the epoch of belief,

it was the epoch of incredulity,

it was the season of Light,

it was the season of Darkness,

it was the spring of hope,

it was the winter of despair,

>>>
```

Notice a couple of things:

1. We are using the "**`with`**" syntax. It does the closing for you, which is nice. In the example, it is the same as: **`f = open("file", "r")`** and then closing the file when you are done.
2. The **`for line in f:`** knows how to read one line at a time (but again, the newlines remain).

Lecture 6: File Processing

If you do want to remove the newlines, you can do it with a list slice:

```
>>> with open("tale_of_two_cities.txt", "r") as f:
...     for line in f:
...         print(line[:-1])
...
It was the best of times,
it was the worst of times,
it was the age of wisdom,
it was the age of foolishness,
it was the epoch of belief,
it was the epoch of incredulity,
it was the season of Light,
it was the season of Darkness,
it was the spring of hope,
it was the winter of despair,
```

You can also read all of the lines into a list, all at once:

```
>>> with open("tale_of_two_cities.txt", "r") as f:
...     all_lines = f.readlines()
...
>>> print(all_lines)
['It was the best of times,\n',
 'it was the worst of times,\n',
 'it was the age of wisdom,\n',
 'it was the age of foolishness,\n',
 'it was the epoch of belief,\n',
 'it was the epoch of incredulity,\n',
 'it was the season of Light,\n',
 'it was the season of Darkness,\n',
 'it was the spring of hope,\n',
 'it was the winter of despair,\n']
```

Once again, notice that the newlines are not removed for you. You could remove them with a list comprehension, like this:

```
>>> with open("tale_of_two_cities.txt", "r") as f:
...     all_lines = [x[:-1] for x in f.readlines()]
```

Lecture 6: Exceptions

Sometimes, your program does not behave the way you want it to, through no real fault of your own. For example, let's say you try to open a file for reading that doesn't exist:

```
>>> with open("my_missing_file.txt", "r") as f:
...     for line in f:
...         print(line)
...
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'my_missing_file.txt'
```

In this case, your program would crash! We can avoid that crash by *catching the exception*, meaning that we have Python tell us that there has been an error. For example:

```
>>> try:
...     with open("my_missing_file.txt", "r") as f:
...         for line in f:
...             print(line)
... except:
...     print("Something went wrong when trying to open the file!")
...
Something went wrong when trying to open the file!
>>>
```

We control the message, and we can recover from the error, instead of crashing the program.

Lecture 6: Exceptions

```
>>> try:
...     with open("my_missing_file.txt", "r") as f:
...         for line in f:
...             print(line)
... except:
...     print("Something went wrong when trying to open the file!")
...
Something went wrong when trying to open the file!
>>>
```

When you have a simple **`except`** statement, this will catch *any* error, which is often not what you want to do. You should try to catch the actual exception that you expect. For example:

```
>>> try:
...     with open("my_missing_file.txt", "r") as f:
...         for line in f:
...             print(line)
... except FileNotFoundError:
...     print("Something went wrong when trying to open the file!")
...
Something went wrong when trying to open the file!
```

We knew that there could be a **`FileNotFoundError`**, so we caught it directly (how did we know? We tried it and saw that error, on the last slide!)

Lecture 6: Exceptions

You can catch any error that the system produces, as long as you know what you are looking for. For example:

```
>>> a = int(input("Please enter an integer: "))
Please enter an integer: October
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'October'
>>> valid_input = False
>>> while not valid_input:
...   try:
...     a = int(input("Please enter an integer: "))
...     valid_input = True
...   except ValueError:
...     print("That wasn't a valid integer...")
...
Please enter an integer: October
That wasn't a valid integer...
Please enter an integer: Bob
That wasn't a valid integer...
Please enter an integer: -3
```

In this case, we continued the program, even though the user kept typing non-integer inputs. We saved ourselves from a crash!