JTSK-350112

Advanced Programming in Python

Python II

Lecture 5 & 6

Dr. Kinga Lipskoch

Spring 2018

Agenda Week 3

- Plotting pixels
- Simulation and pseudo-random numbers
- Top-down design
- Unit testing
- Data plotting using `gnuplot`
- Processing CSV data

Plotting Pixels: `plotPixelFast()` vs. `plotPixel()`

```python
import time
from graphics import *

win = GraphWin("plotPixelFast vs. plotPixel", 300, 300)
message = Text(Point(150, 280), "Click for first pixel")
message.draw(win)
win.getMouse()

start1 = time.time()
for i in range(1000):
    win.plotPixelFast(50, 50, 'blue')
stop1 = time.time()
print(stop1 - start1)

message.setText("Click for second pixel.")
win.getMouse()
start2 = time.time()
for i in range(1000):
    win.plotPixel(100, 100, 'red')
stop2 = time.time()
print(stop2 - start2)

message.setText("Click for exit.")
win.getMouse()
win.close()
```

plotpixels.py

Simulation

- Model describes a real-world process or system
- By solving equations or by a numerical approach the behavior of systems can be investigated

A Simulation of Racquetball

- Bob often plays racquetball with players who are slightly better than he is
- Bob usually loses his matches
- Should not players who are a little better win just a little more often?
- Simulate the game to find out whether only slight differences cause high score differences

Modelling the Game

- Racquetball is played between two players using a racquet to hit a ball in a four-walled court
- One player starts the game by putting the ball in motion - serving
- Players try to alternate hitting the ball to keep it in play, referred to as a rally
- The rally ends when one player fails to hit a legal shot

What is Actually Needed?

- Two players
- One of the players starts the serving (in the simulation `playerA`)
- Only serving winner gets a point
- If the serving player cannot score then serving switches
- Game ends after 15 points
- Other details of game are not important at all

How to Simulate the Different Levels of the Players

- Probability to win his/her own serve
- A probability to of 0.60 means that 60% of serves will be won by serving player
- Needs to be easily configurable to allow different games input values at startup
- Several games should be simulated
- At the end the statistics of the games played should be printed

How to Simulate

- As in the real world, winning 50% of the games does not mean that players take turns as A, then B, then A, ...
- Each game is seen as an independent event
- So previous games should not affect the current game, but the statistics will yield 50% loss and 50% win
- As in a coin toss, the outcome is random, but the overall probability is 50%

Pseudo-random Numbers (1)

- As in previous examples random numbers are needed to simulate a coin toss or to simulate a game of raquetball
- Pseudo-random number generators (RNG) actually create a sequence of numbers that have a very long periodicity
- A seed is being used to initialize the work by starting with a seed value
- The seed basically jumps to a certain position in the sequence
- Therefore by using the same seed again, the same sequence of random numbers will be generated again

Pseudo-random Numbers (2)

- These RNGs are functions which derive an initial seed value from the computer's date and time when the module is loaded, so each time a program is run a different sequence of random numbers is produced
- The two functions of greatest interest to us are `randrange()` and `random()`

Generating Pseudo-random Integers (1)

- The `randrange()` function is used to select a pseudo-random int from a given range
- The syntax is similar to that of the `range()` command
- `randrange(1, 6)` returns some number from `[1, 2, 3, 4, 5]` and `randrange(5, 105, 5)` returns a multiple of 5 between 5 and 100, inclusive
- Ranges go up to, but does not include, the stopping value

Generating Pseudo-random Integers (2)

- Each call of `randrange()` generates a new pseudorandom `int`

```
>>> from random import randrange
>>> randrange(1,6)
5
>>> randrange(1,6)
3
>>> randrange(1,6)
2
>>> randrange(1,6)
5
>>> randrange(1,6)
5
>>> randrange(1,6)
5
```

Generating Pseudo-random Integers (3)

- The value 5 comes up over half the time, demonstrating the probabilistic nature of pseudo-random numbers
- Over time, this function will produce a uniform distribution, which means that all values will appear an approximately equal number of times

Generating Pseudo-random Floating Point Values (1)

- The `random()` function is used to generate pseudo-random floating point values
- It takes no parameters and returns values uniformly distributed between 0 and 1 (including 0 but excluding 1)

Generating Pseudo-random Floating Point Values (2)

```
>>> from random import random
>>> random()
0.7943280091289816
>>> random()
0.00049858619405451776
>>> random()
0.134123140081687 8
>>> random()
0.9872455453536165 3
>>> random()
0.2142942417503219 7
>>> random()
0.2390358371212714 1
>>> random()
0.7291832884340891 9
```

Generating Pseudo-random Floating Point Values (3)

- Assume we generate a random number between 0 and 1
- Exactly 70% of the interval $[0, 1)$ is to the left of 0.7
- So, 70% of the time the random number will be < 0.7, and it will be ≥ 0.7 the other 30% of the time
- The $==$ goes on the upper end since the random number generator can produce a 0 but not a 1

Back to Racquetball

- If `prob` represents the probability of winning the serve, the condition `random() < prob` will succeed with the correct probability
-
```
if random() < prob:
    score = score + 1
```

Top-Down Design

- In top-down design, a complex problem is expressed as a solution in terms of smaller, simpler problems
- These smaller problems are then solved by expressing them in terms of smaller, simpler problems
- This continues until the problems are trivial to solve
- The little pieces are then put back together as a solution to the original problem

Top-Level Design (1)

- Typically a program has the following pattern:
  - Input
  - Process
  - Output
- The algorithm for the racquetball simulation:
  - Print an introduction
  - Get the inputs: `probA`, `probB`, `n`
  - Simulate `n` games of racquetball using `probA` and `probB`
  - Print a report on the wins for `playerA` and `playerB`
- Whatever is not known yet in detail, it will be ignored for now
- Insert high-level functions, details will be done at a later stage

Top-Level Design (2)

- First an introduction should be printed on the screen
- But we do not care about details at this level
- ```
  def main():
      printIntro()
  ```
- We just assume that there is a `printIntro()` function that prints the instructions of the game
  ```

Top-Level Design (3)

- The next step is to get the inputs
- We have already done this several times
- But for now, we again assume that this is already done
- It is important what this function returns the inputted values
- `getInputs()` returns the values for `probA`, `probB`, and `n`
- ```
  def main():
    printIntro()
    probA, probB, n = getInputs()
  ```
  ```

Top-Level Design (4)

- Now we need to simulate n games of racquetball using the values of `probA` and `probB`
- Again a function with a meaningful name should take care of this:
  `simNGames()`
- What parameters does `simNGames()` need?
- What should the function return?

Top-Level Design (5)

- If you were going to simulate the game by hand, what inputs would you need?
  - `probA`
  - `probB`
  - `n`
- What values would you need to get back?
  - The number of games won by player `A`
  - The number of games won by player `B`
- These must be the outputs (return values) of the `simNGames()` function

Top-Level Design (6)

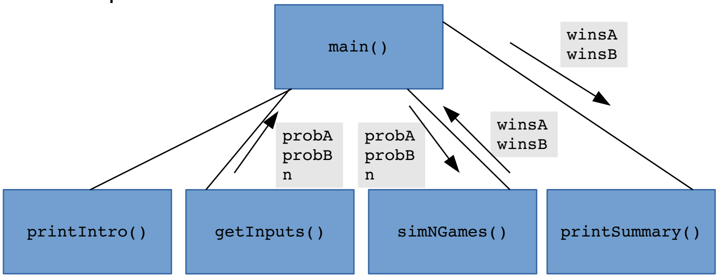
- `winsA, winsB = simNGames(n, probA, probB)`
- A summary at end of games should also be printed: `printSummary()`
- Now the structure of the program is:
-
```
def main():
    printIntro()
    probA, probB, n = getInputs()
    winsA, winsB = simNGames(n, probA, probB)
    printSummary(winsA, winsB)
```

Task Decomposition (1)

- The original problem has now been decomposed into four independent tasks:
  - `printIntro()`
  - `getInputs()`
  - `simNGames()`
  - `printSummary()`
- We did not care about the details, but determined the parameters and return values of the functions
- We have determined the interface or signature of the functions which allows us to complete each single task independently

Task Decomposition (2)

- In a structure chart (or module hierarchy), each component in the design is a rectangle
- A line connecting two rectangles indicates that the one above uses the one below
- The arrows and annotations describes the interfaces between the components



Task Decomposition (3)

- At each level of design, the interface tells us which details of the lower level are important
- The general process of determining the important characteristics of something and ignoring other details is called abstraction
- The top-down design process is a systematic method for discovering useful abstractions

Go to the Next Level

- The next step is to repeat the process for each of the modules defined in the previous step
- The `printIntro()` function should print an introduction to the program

```
def printIntro():
  # Prints an introduction to the program
  print("This program simulates ...")
  print("...")
  print("...")
  print("...")
```

Second-Level Design

In `getInputs()`, we prompt for and get three values, which are converted and returned to the `main()` function

```
def getInputs():
  # RETURNS probA, probB, number of games to simulate
  a = float(input("Prob. of player A to win a serve? "))
  b = float(input("Prob. of player B to win a serve?"))
  n = int(input("How many games to simulate? "))
  return a, b, n
```

Designing `simNGames()` (1)

- This function simulates `n` games and keeps track of how many wins there are for each player
- Simulate `n` games with:
  - Counted loop
  - Tracking wins in some variables
- Pseudocode:

```
Initialize winsA and winsB to 0
loop n times
  simulate a game
  if playerA wins
    Add one to winsA
  else
    Add one to winsB
```

Designing simNGames() (2)

It is easy to get started:

```
def simNGames(n, probA, probB):
  # Simulates n games of racquetball between players A and B
  # RETURNS number of wins for A, number of wins for B
  winsA = 0
  winsB = 0
  for i in range(n):
    ...
```

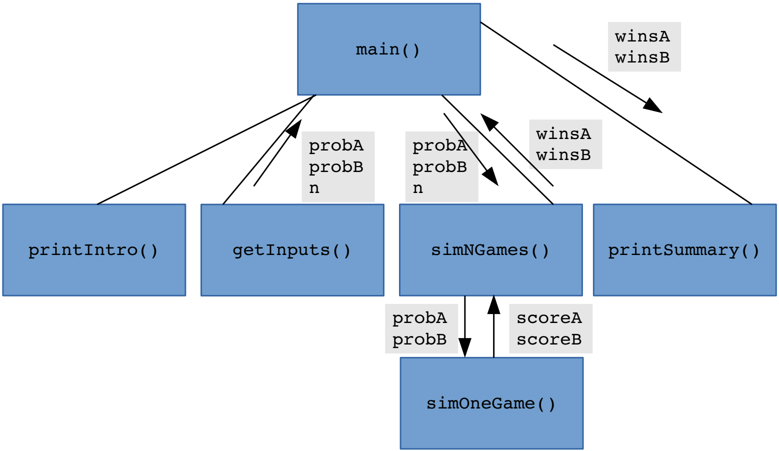
Designing `simNGames()` (3)

- The next thing we need to do is simulate a game of racquetball `simOneGame()`
- The inputs to `simOneGame()` are easy: the probabilities for each player
- What needs to be returned?
- We need to know who won the game
- How can we get this information?
- The easiest way is to pass back the final score
- The player with the higher score wins and gets their win counter incremented by one

Designing `simNGames()` (4)

```python
def simNGames(n, probA, probB):
  # Simulates n games of racquetball between
     players A and B
  # RETURNS number of wins for A, number of wins
     for B
  winsA = winsB = 0
  for i in range(n):
    scoreA, scoreB = simOneGame(probA, probB)
    if scoreA > scoreB:
      winsA = winsA + 1
    else:
      winsB = winsB + 1
  return winsA, winsB
```

Designing `simNGames()` (5)



Third-Level Design (1)

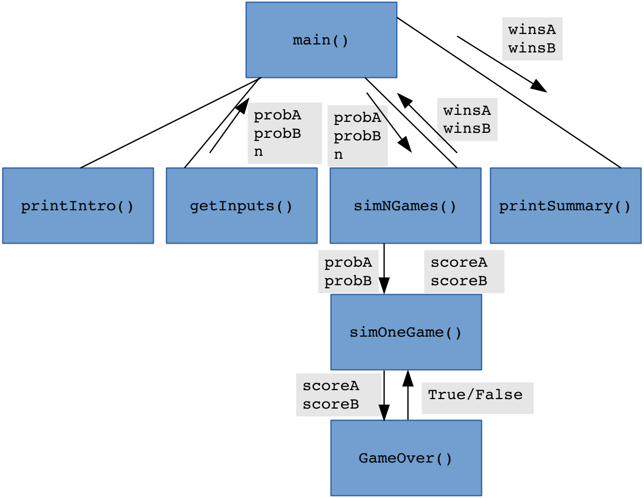
- The next function we need to write is `simOneGame()`, where the logic of the racquetball rules lies
- Players keep doing rallies until the game is over
  - Indefinite loop
  - We do not know in advance when game will be over
  - Break on certain condition
- Keep track of the score
- Keep track of who is serving
- There is no need to use integers for that
- String variable that alternates between "`A`" and "`B`" makes code much more readable

Third-Level Design (2)

Pseudocode:

```
Initialize scores to 0
Set serving to "A"
Loop while game is not over:
  Simulate one serve of whichever player is
    serving
  Update the status of the game
Return scores
```

Third-Level Design (3)



Third-Level Design (4)

At this point, `simOneGame()` looks like this:

```
def simOneGame(probA, probB):
  # Simulates a single game or racquetball
    between players A and B
  # RETURNS A's final score, B's final score
    serving = "A"
    scoreA = 0
    scoreB = 0
    while not gameOver(scoreA, scoreB):
      ...
```

Third-Level Design (5)

- Inside the loop:
  - Single serve
  - Compare random number against prob to determine who has won (`random() < prob`)
- Probability to use is determined by whom is serving, contained in the variable serving
- If `A` is serving, then we use `A`'s probability, and based on the result of the serve, either update `A`'s score or change the service to `B`

```
if serving == "A":
  if random() < probA:
    scoreA = scoreA + 1
  else:
    serving = "B"
```

Third-Level Design (6)

Likewise, if it is `B`'s serve, we will do the same thing with a mirror image of the code

```
if serving == "A":
  if random() < probA:
    scoreA = scoreA + 1
  else:
    serving = "B"
else:
  if random() < probB:
    scoreB = scoreB + 1
  else:
    serving = "A"
```

Complete simOneGame()

```
def simOneGame(probA, probB):
  # Simulates a single game or racquetball between players A
     and B
  # RETURNS A's final score, B's final score
  serving = "A"
  scoreA = 0
  scoreB = 0
  while not gameOver(scoreA, scoreB):
    if serving == "A":
      if random() < probA:
        scoreA = scoreA + 1
      else:
        serving = "B"
    else:
      if random() < probB:
        scoreB = scoreB + 1
      else:
        serving = "A"
  return scoreA, scoreB
```

Finishing Up

```python
def gameOver(a, b):
    # a and b are scores for players in a
    racquetball game
    # RETURNS True if game is over, False
    otherwise
    return a == 15 or b == 15
```

```python
def printSummary(winsA, winsB):
    # Prints a summary of wins for each player
    n = winsA + winsB
    print("\nGames simulated:", n)
    print("Wins for A: {0} ({1:0.1%})".format(
    winsA, winsA/n))
    print("Wins for B: {0} ({1:0.1%})".format(
    winsB, winsB/n))
```

Summary of the Design Process

- We started at the highest level of our structure chart and worked our way down
- At each level, we began with a general algorithm and refined it into precise code
- This process is sometimes referred to as step-wise refinement:
  1. Express the algorithm as a series of smaller problems
  2. Develop an interface for each of the small problems
  3. Detail the algorithm by expressing it in terms of its interfaces with the smaller problems
  4. Repeat the process for each smaller problem

Unit Testing (1)

- A good way to systematically test the implementation of a modestly sized program is to start at the lowest levels of the structure, testing each component as it is completed
- For example, we can import our program and execute various routines/functions to ensure they work properly
- We could start with the `gameOver()` function

```
>>> import rball
>>> rball.gameOver(0,0)
False
>>> rball.gameOver(5,10)
False
>>> rball.gameOver(15,3)
True
>>> rball.gameOver(3,15)
True
```

Unit Testing (2)

- Notice that we have tested `gameOver()` for all the important cases
- We gave 0, 0 as inputs to simulate the first time the function will be called
- The second test is in the middle of the game, and the function correctly reports that the game is not yet over
- The last two cases test to see what is reported when one of the players has won

Unit Testing (3)

- Now, we can test `simOneGame()`

```
>>> simOneGame(.5, .5)
(11, 15)
>>> simOneGame(.5, .5)
(13, 15)
>>> simOneGame(.3, .3)
(11, 15)
>>> simOneGame(.3, .3)
(15, 4)
>>> simOneGame(.4, .9)
(2, 15)
>>> simOneGame(.4, .9)
(1, 15)
>>> simOneGame(.9, .4)
(15, 0)
>>> simOneGame(.9, .4)
(15, 0)
>>> simOneGame(.4, .6)
(10, 15)
```

Unit Testing (4)

- When the probabilities are equal, the scores are not that far apart
- When the probabilities are farther apart, the game is a rout for one of the players
- Testing each component in this manner is called unit testing
- Testing each function independently makes it easier to spot errors, and should make testing of the entire program go more smoothly

Simulation Results (1)

- Is it the nature of racquetball that small differences in ability lead to large differences in final score?
- Suppose Bob wins about 60% of his serves and his opponent is 5% better
- How often should Bob win?
- Let's do a sample run where Bob opponent serves first

Simulation Results (2)

```
This program simulates a game of racquetball
between two players called "A" and "B".
The abilities of each player is indicated by a
probability (a number between 0 and 1) that the
player wins the point when serving. Player A
always has the first serve.

Prob. of player A to win a serve? .65
Prob. of player B to win a serve? .6
How many games to simulate? 5000

Games simulated: 5000
Wins for A: 3329 (66.6%)
Wins for B: 1671 (33.4%)
```

With this small difference in ability, Bob will win only 1 in 3 games

CSV Data Format (1)

- CSV: comma separated values: "`Mount Everest`", 8848, 1953
- Very simple, easy to use format for data exchange
- Safest format for data exchange is XML, will not be covered in this course

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

CSV Data Format (2)

1. One record per line
2. Each record has several fields which are separated by commas
3. Field is either string or number
4. Strings are enclosed in either single or double quotes
5. Commas are allowed inside strings and must not be treated as field separators

Visualizing Data (1)

- We want to visualize weather data of 61169 Friedberg, Germany in 2008
- Why Friedberg?
  - Because the data of their weather station was freely accessible (not anymore)
  - http://wetter61169.de/download/
  - At the moment accessible at https://grader.eecs.jacobs-university.de/courses/350112/python/csv/
  - Data for January https://grader.eecs.jacobs-university.de/courses/350112/python/csv/exp200801.csv
- Why 2008?
  - Because they switched from `.csv` to `.xls`
  - We could handle `.xls` as well, but `.csv` is easier

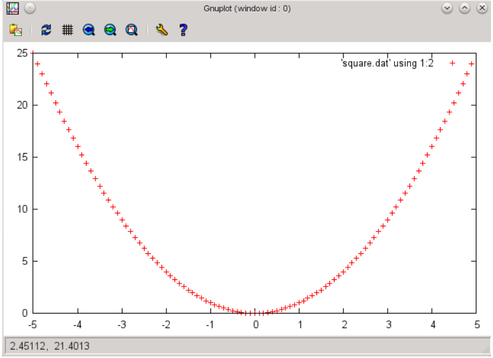
Visualizing Data (2)

- We need to download CSV files from their site
  - Modules `urllib`, `urllib.request`
- We need to parse the CSV file and change the data layout according to our needs
  - Module `csv`
- How to visualize?
  - Modules `numpy`, `matplotlib`
  - Unfortunately not yet included in python3

Visualizing Data (3)

- Find other ways to display data
- Use `graphics.py` to plot data
  - Set the right coordinates and plot the data
- Use other programs and/or modules
- Use `gnuplot`
  - Available on all platforms,
  - Plots data from a textfile
  - Simple `x y1 y2 y3` data format
  - Short introduction for a few details: http://people.duke.edu/~hpgavin/gnuplot.html

Using gnuplot



square.dat

```
-5.00 25.00
-4.90 24.01
-4.80 23.04
-4.70 22.09
-4.60 21.16
-4.50 20.25
-4.40 19.36
-4.30 18.49
-4.20 17.64
-4.10 16.81
-4.00 16.00
-3.90 15.21
-3.80 14.44
...
```

```
gnuplot> plot 'square.dat' using 1:2
```

Use gnuplot from Python (1)

```
import subprocess
proc = subprocess.Popen(['gnuplot'], shell =
    True, stdin = subprocess.PIPE)

proc.stdin.write(b'set term png\n')
proc.stdin.write(b'set output "out.png"\n')
proc.stdin.write(b'plot
      "squares.dat" using 1:2\n')
proc.stdin.write(b'quit\n')
proc.stdin.flush()
```

plot.py

squares.dat

Use gnuplot from Python (2)

- For Windows:
  - Need to define environment variables for the installation directories of Python and gnuplot
  - Add those variables to the environment variable `PATH`
  - You can do this using Control Panel
- Other plattforms: install gnuplot (if not already installed)
- If because of some reason you cannot install, at least use online version to check results and generate plots
  http://gnuplot.respawned.com/

Plotting Weather Data using gnuplot

Instead of writing a CSV file write to a file that contains spaces as column delimiters

```
gnuplot> set timefmt "%Y-%m-%d %H:%M:%S"
gnuplot> set xdata time
gnuplot> set format x "%H:%M"
gnuplot> plot 'somedata.dat' using 1:3
```

somedata.dat

URL

- Uniform Resource Locator (URL)
- https://grader.eecs.jacobs-university.de/courses/350112/python/csv/exp200801.csv
- `https://` Scheme/Protocol
- `grader.eecs.jacobs-university.de` Host
- `/courses/350112/python/csv/exp200801.csv` Path/filename
- More components exist, but not relevant here

Reading an URL

```
import sys
import urllib.request
url = 'https://grader.eecs.jacobs-university.de/courses/350112/python/csv/exp200801.csv'

try:
  u = urllib.request.urlopen(url)
except:
  print("Error fetching URL: ", url)
  sys.exit(1)
lines = u.readlines()
# print content to standard output
for element in lines:
  print(element)
```

read_url.py

Retrieving a URL

```
import sys
import urllib.request

url = 'https://grader.eecs.jacobs-university.de/
    courses/350112/python/csv/exp200801.csv'
outfile = 'weather.dat'

try:
  u = urllib.request.urlretrieve(url, outfile)
except:
  print("Error fetching URL: ", url)
  sys.exit(1)
```

retrieve_url.py

weather.dat

Weather Data as CSV

```
"Station Name","ST JOHN'S A"
"Province","NEWFOUNDLAND"
"Latitude","47.62"
"Longitude","-52.74"
"Elevation","140.50"
"Climate Identifier","8403506"
"WMO Identifier","71801"
"TC Identifier","YYT"

"All times are specified in Local Standard Time (LST). Add 1 hour to adjust for
Daylight Saving Time where and when it is observed."

"Legend"
"M","Missing"
"E","Estimated"
"NA","Not Available"

"Date/Time","Year","Month","Day","Time","Temp (C)","Temp Flag","Dew Point Temp
(C)","Dew Point Temp Flag","Rel Hum (%)","Rel Hum Flag","Wind Dir (10's deg)","Wind
Dir Flag","Wind Spd (km/h)","Wind Spd Flag","Visibility (km)","Visibility Flag","Stn
Press (kPa)","Stn Press Flag","Hmdx","Hmdx Flag","Wind Chill","Wind Chill
Flag","Weather"
"2007-10-1
0:00","2007","10","1","0:00","","","","","","","","","","","","","","","","","","",""
"2007-10-1
0:30","2007","10","1","0:30","2.10","","0.60","","90.00","","28.00","","13.00","","24
.10","","101.22","","","","","","Clear"
"2007-10-1
1:00","2007","10","1","1:00","","","","","","","","","","","","","","","","","","",""
```

Example How to Read such a File (1)

```
import csv
import datetime

def extract_time(row):
  y = int(row[1])
  mn = int(row[2])
  d = int(row[3] )
  h,m = row[4].split(':')
  h = int(h)
  m = int(m)
  return datetime.datetime(y, mn, d, h, m)

def extract_temp(row):
  t = row[5]
  try:
    return float(t)
  except:
    return None
```

Example How to Read such a File (2)

- `reader.py` is being used to clean up data from a Canadian source
- You will need to adapt this file to your specific needs
- `oct.csv` is an example file for processing

Example How to Read such a File (3)

```
def extract_rel_humidity(row):
  rh = row[9]
  try:
    return float(rh)
  except:
    return None

def extract_wind_dir(row):
  wd = row[11]
  try:
    return float(wd) * 10.0
  except:
    return None

def extract_wind_speed(row):
  sp = row[13]
  try:
    return float(sp)
  except:
    return None
```

Example How to Read such a File (4)

```python
def extract_pressure(row):
  p = row[17]
  try:
    return float(p)
  except:
    return None

def extract_description(row):
  return row[23]

f = open("oct.csv")
reader = csv.reader(f)
name = next(reader)[0]
prov = next(reader)[0]
latitude = next(reader)[0]
longitude = next(reader)[0]
elevation = next(reader)[0]
```

Rewriting the CSV File (1)

```python
# skip to line 16 the line number of the input
    file is maintained in line_num
while reader.line_num < 16:
  next(reader)

headers = next(reader)
print(headers)

fout = open("reduced.csv", "w")
writer = csv.writer(fout)

writer.writerow(('time', 'temp', 'humidity',
  'wind dir', 'wind speed', 'pressure',
  'desc'))
```

Rewriting the CSV File (2)

```
for row in reader:
  t = extract_time(row)
  if not t:
    continue
  temp = extract_temp(row)
  if not temp:
    continue
  h = extract_rel_humidity(row)
  if not h:
    continue
  wd = extract_wind_dir(row)
  if not wd:
    continue
  ws = extract_wind_speed(row)
  if not ws:
    continue
  press = extract_pressure(row)
  if not press:
    continue
  description = extract_description(row)
  if not description:
    continue
  writer.writerow((t, temp, h, wd, ws, press, description))
fout.close()
f.close()
```

Regular Expressions

- Powerful way to search text for strings or patterns
- Powerful way to replace strings using patterns
- `re.findall(r, s)`
  - Returns all nonoverlapping matches if the regular expression `r` matches in string `s`
- `re.sub(r, x, s)`
  - Returns a copy of string `s` with every match of `r` replaced with the string `x`
- A few examples:
  - `regexp_search.py`
  - `regexp_replace.py`

Final Exam

- Thursday, 24th of May, 2018
- 12:30 - 14:30 in East Wing, IRC
- Written on paper
- Similar problems as in the assignments covering most of the important discussed topics
- Practice sheet on the webpage of the course: Practice sheet
- Tutorial given by TAs a few days before the exam