DATA 520
Lecture 17

Homework 11

Introduction to Algorithms

Homework 11

Due 10/25 before class

Exercises 10.10

1, 2,

A. Using for and while, write a function that will warn if a file exists, ask the user if he/she wants to choose another name, overwrite, or simply cancel. Of course, any new name must be tested too (while). You have most of the necessary code on slide 7 and 8. This will be part of your toolkit.

and...

Homework 11 continued:

B. Read one format into another

Hanihara data: (save to a text file)

```
Specimen 1
  182.00  179.00  100.00  129.00   95.00  108.00  115.00  114.00  100.00  132.00
  130.00  134.00  103.00  113.00  120.00   88.00  105.00  107.00  125.00   94.00
   65.00   23.00   44.00   41.00   37.00   27.00   50.00   49.00   71.00   31.00
   20.00  109.00   83.00  110.00   30.00   12.00   35.00   61.00   35.00  101.00
   54.00   54.00   11.12    5.70    6.89   95.00   52.00   54.00
Specimen 2
  174.00  172.00   96.00  124.00   95.00  110.00  104.00  103.00   96.00  137.00
  127.00  125.00  108.00  112.00  112.00   95.00   93.00  102.00    0.00   86.00
   61.00   21.00   41.00   39.00   35.00   25.00   50.00   49.00   62.00   25.00
   17.00   99.00   83.00  100.00   27.00   11.00   30.00   50.00   31.00   94.00
   49.00   51.00    8.73    0.00    0.00   84.00   47.00   47.00
Specimen 3
  170.00  167.00   92.00  130.00   93.00  109.00  116.00  115.50  100.00  130.00
  123.00  123.00  110.00  107.00  109.00   92.00   93.00  102.00  125.00   87.00
   65.00   21.00   40.00   38.00   34.00   24.00   45.00   45.00   61.00   19.00
   13.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   91.60
   49.10   48.70    7.76    5.22    3.97   89.20   48.60   49.70
```

Convert to .csv format

Read one format into another

Hanihara data format:

Specimen 1 <- specimen number always less than 30 characters.

48 measurements (mostly integers)

0.00 = missing (change to 'NA')

The first line of the converted .csv file will be (all one line):

```
"HSpecNo","GOL","NOL","BNL","XCB","M9","XFB","M11","AUB","ASB","BBH","M26","M27","M28","FRC",
"PAC","OCC","BPL","M43","ZYB","M46","NPH","DKB","M51","OBH","OBH","NLB","NLH","M55","MAB",
"MDH","MDB","U1","U2","U3","U4","U5","U6","U7","U8","U9","U10","U11","WNB","SIS","U12","ZMB",
"U13","U14"
```

- then append the data from the records into the file delimited using commas.

File example:

```
"HSpecNo","GOL","NOL","BNL","XCB"," ...
"Specimen 1",182.00,179.00, 100.00, 129.00, 95.00, 108.00, ...
```

Submit code and file. Think about helper functions (part of a toolkit).

Read one format into another: pseudocode

```
1. Hanihara data format: specimen number on one line; then 48 measurements over 5 lines, space-separated;
2. open new csv file for writing (writefile)
3. write field names (fixed field names) delimited by commas ("HSpecNo","GOL","NOL", ...) to writefile
4. open hanihara.txt file for reading (readfile)
5. read one line from readfile, it is Specimen number, into string variable (newline), add comma
6. for next 5 lines:
     read all values from line into variable # ambiguous
     if any values are 0.00, change to 'NA'
     write all values to newline, but separated by commas
     write newline to writefile
7. loop back to 5 until EOF (end of file)
8. close writefile (if necessary)
9. close readfile (if necessary)
```

Read one format into another: pseudocode

```
1. Hanihara data format: specimen number on one line; then 48 measurements over 5 lines, space-separated;
2. open new csv file for writing (writefile)
3. write field names (fixed field names) delimited by commas ("HSpecNo","GOL","NOL", ...) to writefile
4. open hanihara.txt file for reading (readfile)
5. read one line from readfile, it is Specimen number, into string variable (newline), add comma
6. for next 5 lines:
    read all values from line into variable (split() line into list of items)
    if any values are 0.00, change to 'NA' (easy to do with lists)
    append all values to newline, separated by commas
    write newline to writefile
    # get rid of extra comma at end of line if possible
7. loop back to 5 until EOF (end of file)
8. close writefile (if necessary)
9. close readfile (if necessary)

# this method appends values to newline; it could also write directly to writefile
```

Read one format into another: pseudocode

Notes:

Always use split when working with values separated by something (comma, space, tab)

```
Hanihara data:
Specimen 1
  182.00  179.00  100.00  129.00   95.00  108.00  115.00  114.00  100.00  132.00
```

What happens if you replace '0.00' in that line?

```
  182.00  179.00  10NA  129.00   95.00  108.00  115.00  114.00  10NA  132.00
```

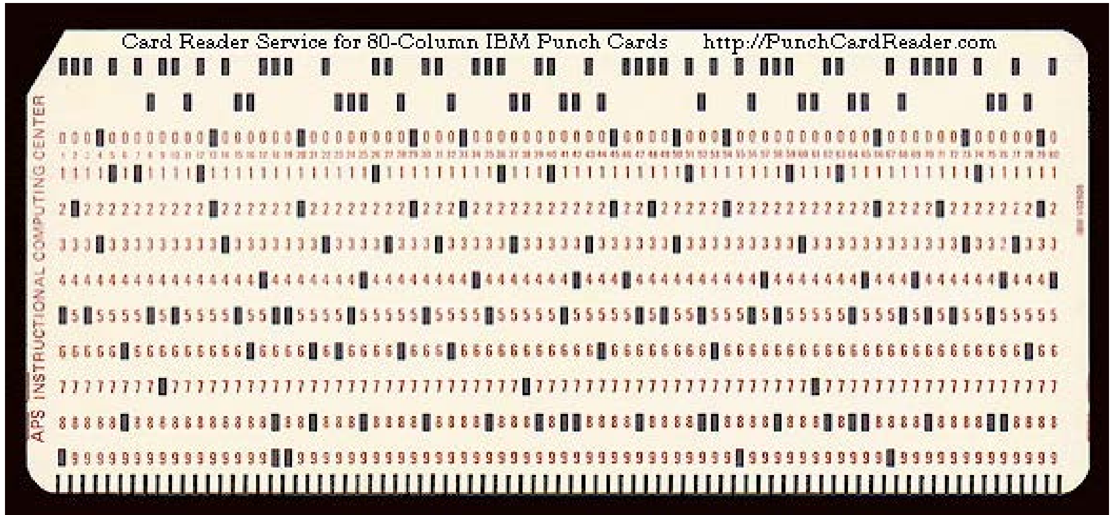
Homework 11

```
def process_file(file, out_file):
    """ (string1, string2) -> write to file string2 """
    with open(file, 'r') as read_file:
        with open(out_file, 'w') as write_file:
            field_list = 
'"HSpecNo","GOL","NOL","BNL","XCB","M9","XFB","M11","AUB","ASB","BBH","M26","M27","M28","FRC","PAC","OCC","BPL",
"M43","ZYB","M46","NPH","DKB","M51","OBH","OBH","NLB","NLH","M55","MAB","MDH","MDB","U1","U2","U3","U4","U5","U6
","U7","U8","U9","U10","U11","WNB","SIS","U12","ZMB","U13","U14"'
            write_file.write(field_list)

            line_number = 0
            for line in read_file:
                write_line = ''
                line = line.strip()
                if 'Specimen' in line: # or length < 30 for more flexibility
                    write_file.write('\n{0},'.format(line))
                else:
                    line_parse = line.split()
                    while '0.00' in line_parse:
                        line_parse[line_parse.index('0.00')] = 'NA'
                    for items in line_parse:
                        write_line += items + ','

                    # keep track of lines written
                    line_number += 1

                    if line_number < 5:
                        write_file.write(write_line)
                    else:
                        # Done, get rid of extra comma at end
                        write_line = write_line[0: len(write_line)-1]
                        # print(write_line)
                        write_file.write(write_line)
                        line_number = 0

if __name__ == '__main__':
    process_file('Hanihara.txt', 'haniharaO.csv')
```

Read a punch card file?

Utermohle data:

```
1AB    432 1       182179 99130132110   11013212176104 997211623634726133655
2AB    432 1       435236420736 99   981510011190805020311227521092351 93314552
3AB    432 1       141161034113 91101    95 82 82 78 72 76
1AB    182 1       194190108143141117   119137128841141097212324625122124054
2AB    182 1       425236430738100211031910312210906020512229601132053104315961
3AB    182 1         1301123913104113103    93 87 85 81 84
1CG2426921 1       182178103139137117   12114213279112 996812521664326133959
2CG2426921 1       50523540094110117 9513 9705201003010311228541102257 97264347
3CG2426921 1       191201034818 94102 97102 86 84 83 78 81
1CG2428341 1       179173102135132115   11914112377112 997212325705028113958
2CG2428341 1       41513340093610122 9913 9907201202010410524471072649 96304753
3CG2428341 1       191151004420 91103 97 99 79 81 78 74 77
```


Card Reader Service for 80-Column IBM Punch Cards
http://PunchCardReader.com
APS INSTRUCTIONAL COMPUTING CENTER

Application Monday: Fordisc

Current version 3.1

Fordisc compares measurements from an unknown individual to those of known individuals to aid in forensic identification.

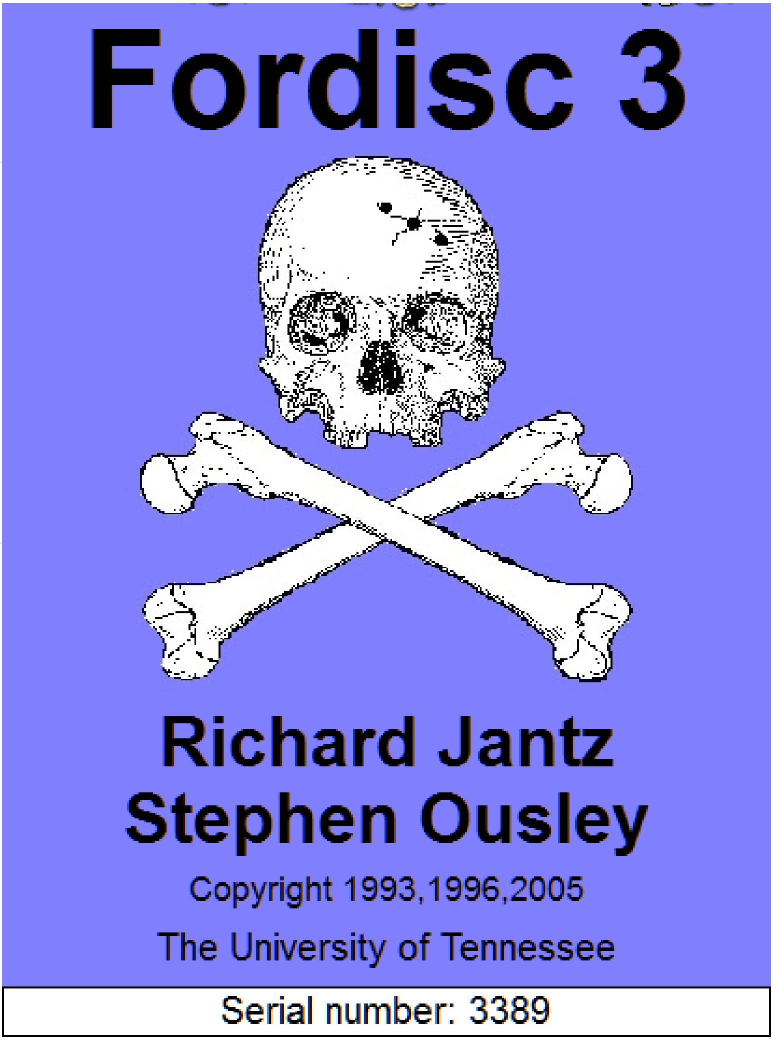
Used by over 600 forensic anthropologists around the world.

Runs on Windows PCs.

Updated regularly.

45 current site licenses around the world (up to 1,000 students every semester).

Extensive help file with tutorial.



Fordisc 3

Richard Jantz
Stephen Ousley

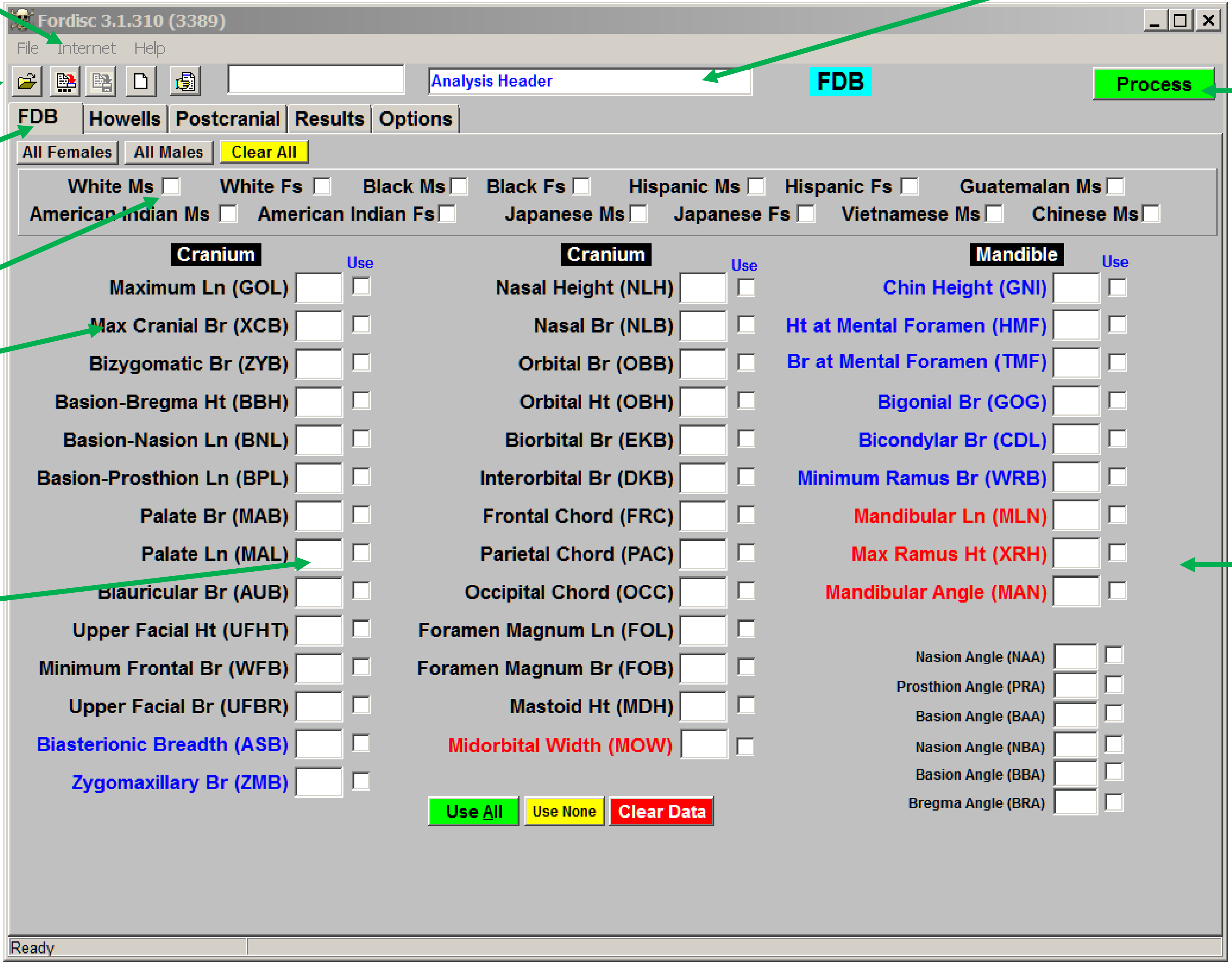
Copyright 1993,1996,2005

The University of Tennessee

Serial number: 3389

Application Monday

Fordisc



White Ms □ White Fs □ Black Ms □ Black Fs □ Hispanic Ms □ Hispanic Fs □ Guatemalan Ms □
American Indian Ms □ American Indian Fs □ Japanese Ms □ Japanese Fs □ Vietnamese Ms □ Chinese Ms □

Cranium		Use	Cranium		Use	Mandible		Use
Maximum Ln (GOL)		□	Nasal Height (NLH)		□	Chin Height (GNI)		□
Max Cranial Br (XCB)		□	Nasal Br (NLB)		□	Ht at Mental Foramen (HMF)		□
Bizygomatic Br (ZYB)		□	Orbital Br (OBB)		□	Br at Mental Foramen (TMF)		□
Basion-Bregma Ht (BBH)		□	Orbital Ht (OBH)		□	Bigonial Br (GOG)		□
Basion-Nasion Ln (BNL)		□	Biorbital Br (EKB)		□	Bicondylar Br (CDL)		□
Basion-Prosthion Ln (BPL)		□	Interorbital Br (DKB)		□	Minimum Ramus Br (WRB)		□
Palate Br (MAB)		□	Frontal Chord (FRC)		□	Mandibular Ln (MLN)		□
Palate Ln (MAL)		□	Parietal Chord (PAC)		□	Max Ramus Ht (XRH)		□
Biauricular Br (AUB)		□	Occipital Chord (OCC)		□	Mandibular Angle (MAN)		□
Upper Facial Ht (UFHT)		□	Foramen Magnum Ln (FOL)		□			
Minimum Frontal Br (WFB)		□	Foramen Magnum Br (FOB)		□	Nasion Angle (NAA)		□
Upper Facial Br (UFBR)		□	Mastoid Ht (MDH)		□	Prosthion Angle (PRA)		□
Biasterionic Breadth (ASB)		□	Midorbital Width (MOW)		□	Basion Angle (BAA)		□
Zygomaxillary Br (ZMB)		□				Nasion Angle (NBA)		□
						Basion Angle (BBA)		□
						Bregma Angle (BRA)		□

menu items

textbox

action button

button

tab

checkbox

text label

Forensic groups craniometrics

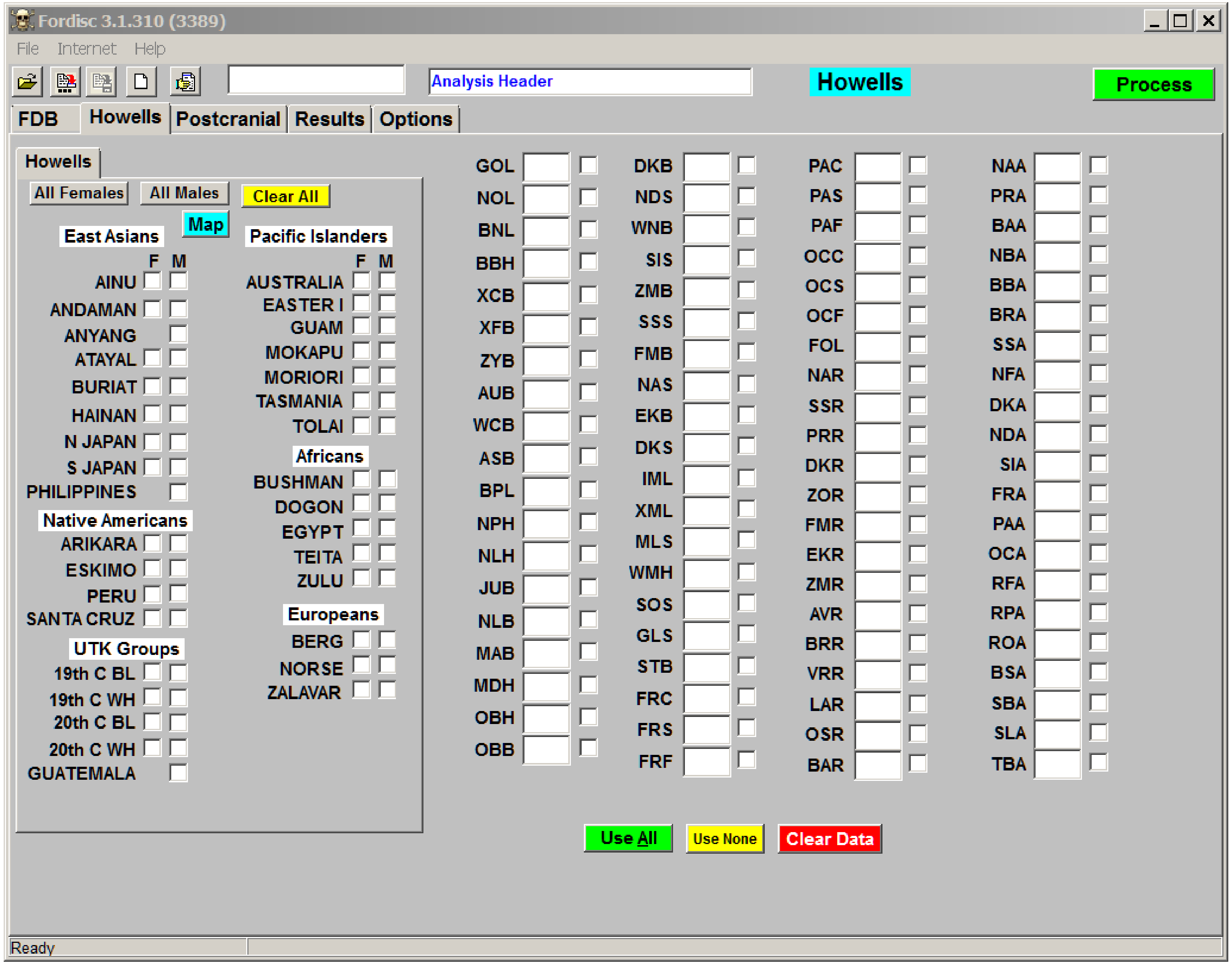
page

Table field text

Forensic groups

Application Monday

Fordisc

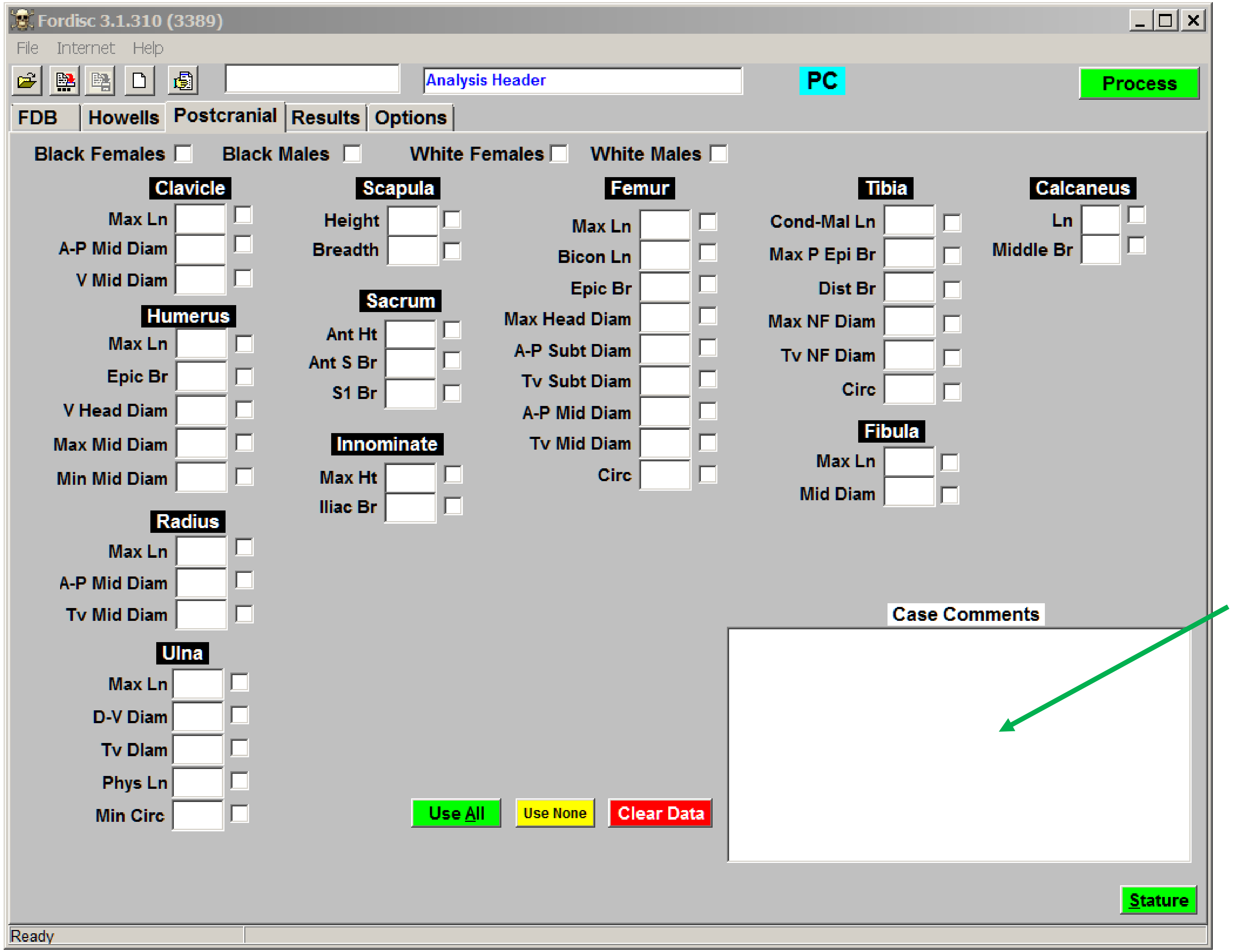


Howells groups craniometrics

Application Monday

Fordisc

Forensic groups postcranial measurements



memo field display

Application Monday

Fordisc

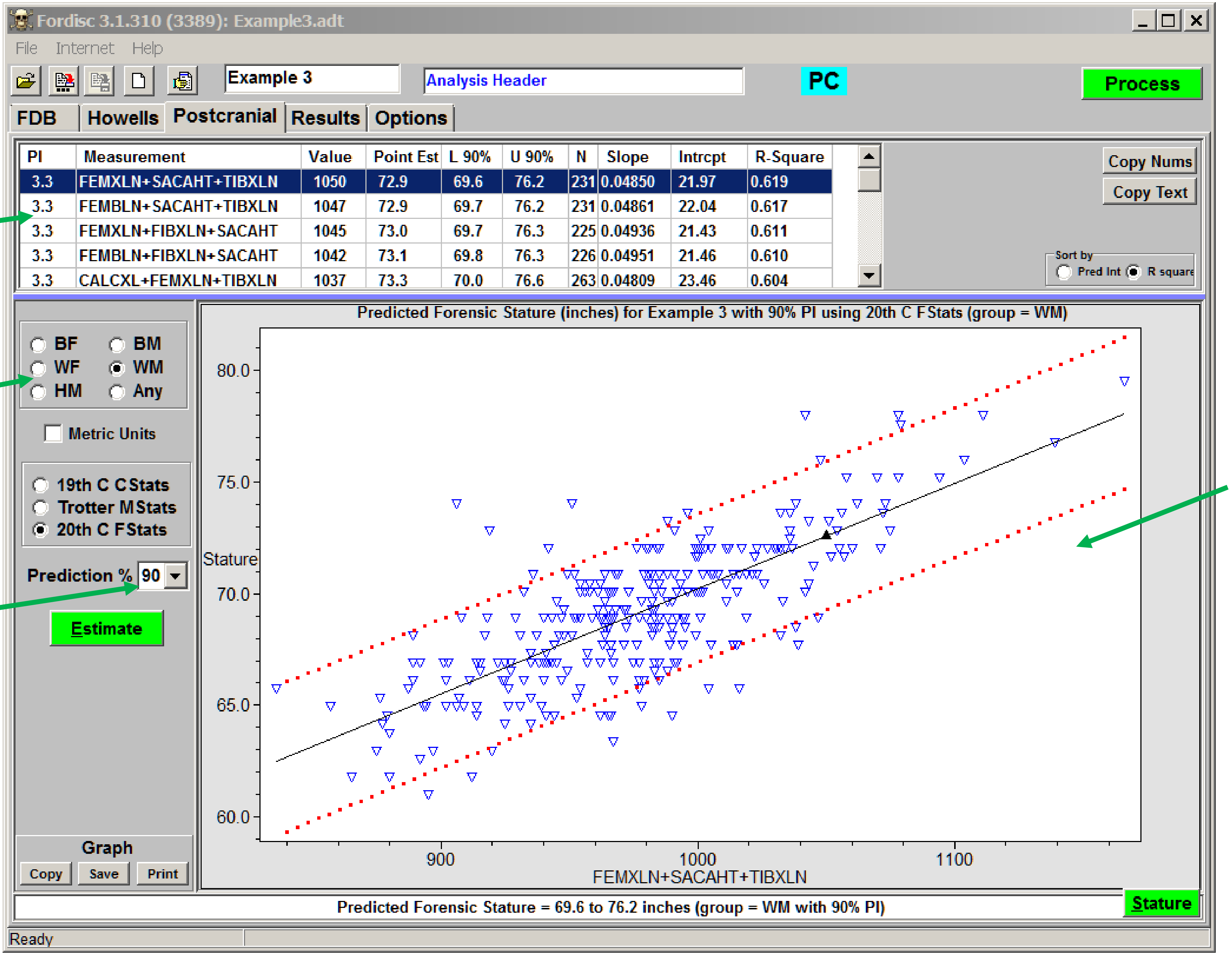
Stature estimation

data grid

radio buttons

drop-down list box

graphics pane



PI	Measurement	Value	Point Est	L 90%	U 90%	N	Slope	Intrcpt	R-Square
3.3	FEMXLN+SACAHT+TIBXLN	1050	72.9	69.6	76.2	231	0.04850	21.97	0.619
3.3	FEMBLN+SACAHT+TIBXLN	1047	72.9	69.7	76.2	231	0.04861	22.04	0.617
3.3	FEMXLN+FIBXLN+SACAHT	1045	73.0	69.7	76.3	225	0.04936	21.43	0.611
3.3	FEMBLN+FIBXLN+SACAHT	1042	73.1	69.8	76.3	226	0.04951	21.46	0.610
3.3	CALCXL+FEMXLN+TIBXLN	1037	73.3	70.0	76.6	263	0.04809	23.46	0.604

Sort by: ○ Pred Int ● R square

○ BF ○ BM
○ WF ● WM
○ HM ○ Any

□ Metric Units

○ 19th C CStats
○ Trotter MStats
● 20th C FStats

Prediction % 90

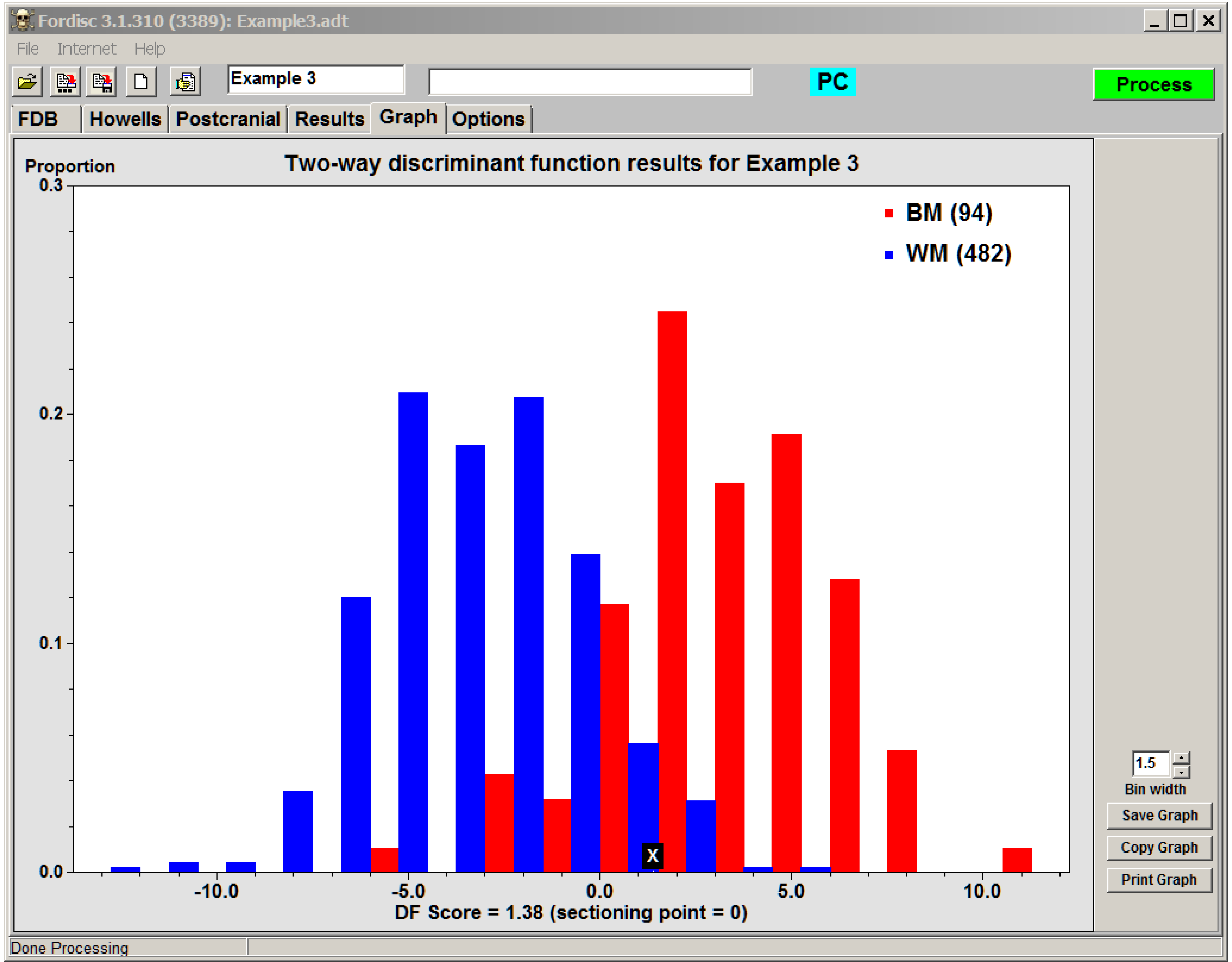
Predicted Forensic Stature (inches) for Example 3 with 90% PI using 20th C FStats (group = WM)

Predicted Forensic Stature = 69.6 to 76.2 inches (group = WM with 90% PI)

Application Monday

Fordisc

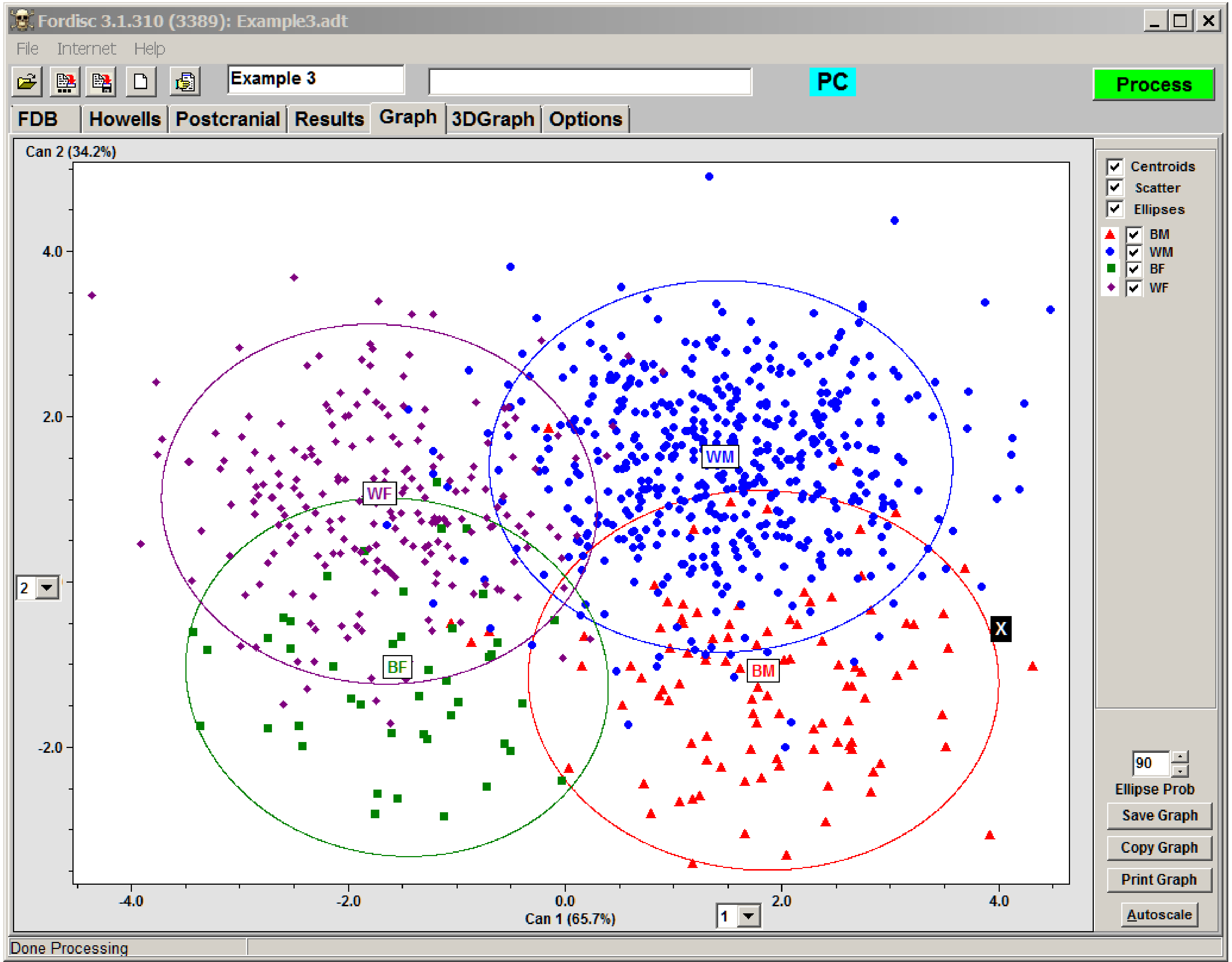
Two-group classification barchart



Application Monday

Fordisc

Multi-group
classification plot

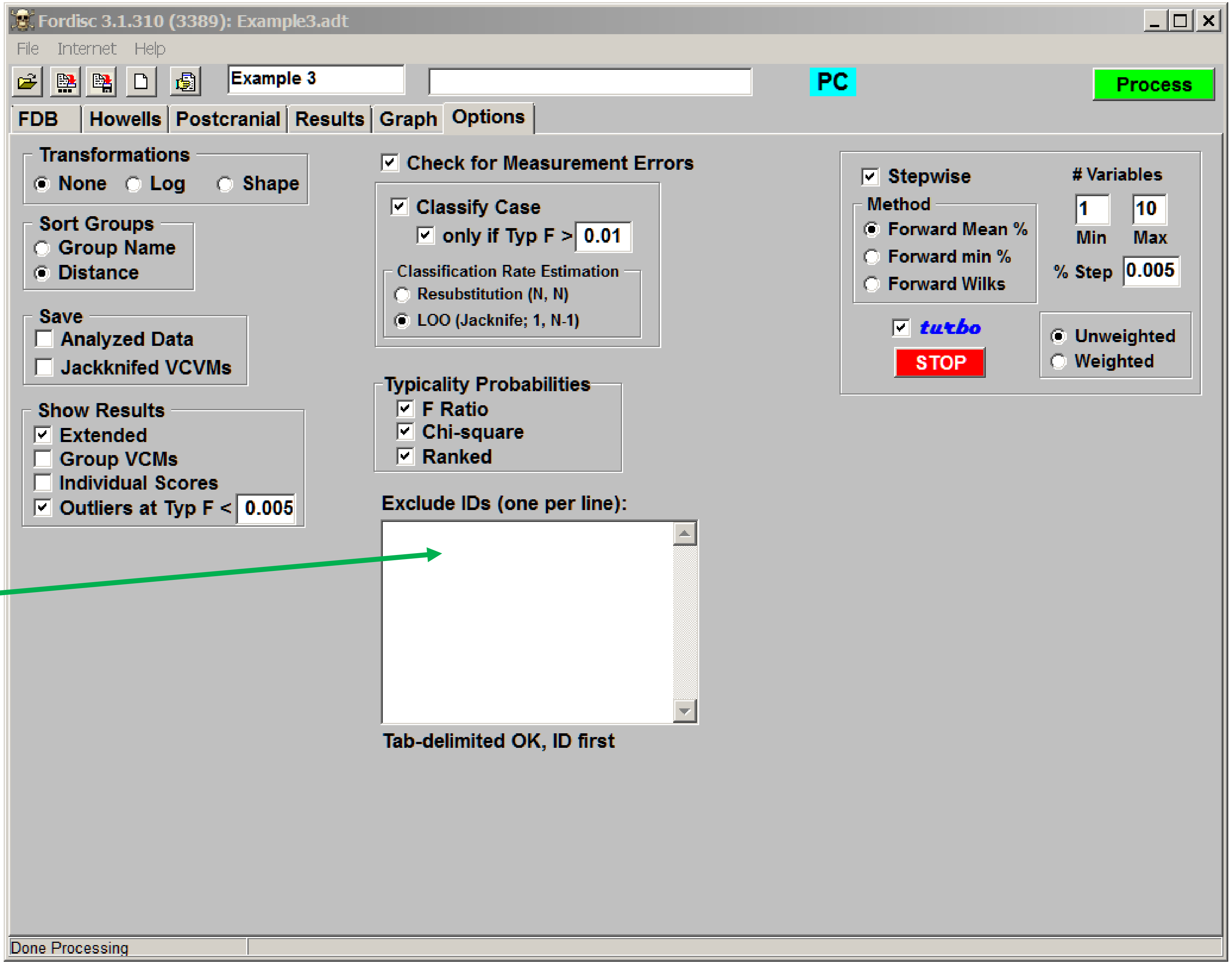


Application Monday

Fordisc

Settings
Options

list box



Transformations
● None ○ Log ○ Shape

Sort Groups
○ Group Name
● Distance

Save
□ Analyzed Data
□ Jackknifed VCVMs

Show Results
☑ Extended
□ Group VCMs
□ Individual Scores
☑ Outliers at Typ F < 0.005

☑ Check for Measurement Errors

☑ Classify Case
☑ only if Typ F > 0.01

Classification Rate Estimation
○ Resubstitution (N, N)
● LOO (Jacknife; 1, N-1)

Typicality Probabilities
☑ F Ratio
☑ Chi-square
☑ Ranked

Exclude IDs (one per line):

Tab-delimited OK, ID first

☑ Stepwise

Method
● Forward Mean %
○ Forward min %
○ Forward Wilks

Variables
Min 1
Max 10
% Step 0.005

☑ turbo

● Unweighted
○ Weighted

Designing Algorithms

Top-down design:

Describe the goal and process in English

Write in pseudocode

- one process per line

Remember/discover Python's built-in functions and methods

- don't forget modules (Google)

(no need to reinvent the wheel!)

Finally, write code in Python

Designing Algorithms

Finding the smallest item in a list already a function min()

```
>>> whalecounts = [809, 834, 477, 478, 307, 122, 96, 102, 324, 476]
>>> min(whalecounts)

# get the index number (verbose)
>>> low = min(whalecounts)
>>> min_index = whalecounts.index(low)
>>> print(min_index) # or >>> min_index

# get the index number (one line)
counts.index(min(counts))
```

Designing Algorithms

How to find the TWO smallest items?

English: Find the two smallest items (their indices) in a list and preserve the list

Pseudocode:

We realize there is more than one way using code

1. find min, remove item, find min from rest (then restore item) - FRF

2. copy list, sort it, get two smallest - STF

3. walk through values and keep track of smallest two - WTV

Our criterion is *speed*! Time needed to perform everything.

Designing Algorithms

1. FRF

```
# find_remove_find5.py
whalecounts = [809, 834, 477, 478, 307, 122, 96, 102, 324, 476]
def find_two_smallest(L):
    """ (list of float) -> tuple of (int, int)
    Return a tuple of the indices of the two smallest values in list L.
    >>> find_two_smallest([809, 834, 477, 478, 307, 122, 96, 102, 324, 476])
    (6, 7)
    """
    # Pseudocode:
    # Find the index of the minimum item in L
    # Remove that item from the list
    # Find the index of the new minimum item in the list
    # Put the smallest item back in the list
    # If necessary, adjust the second index
    # Return the two indices of the smallest
```

Designing Algorithms

1. FRF - body - add to rest of `find_remove_find5.py`

```
# Find the index of the minimum and remove that item
smallest = min(L)
min1 = L.index(smallest)
L.remove(smallest)
# Find the index of the new minimum
next_smallest = min(L)
min2 = L.index(next_smallest) # min2 index may have changed
# Put the smallest item back in the list
L.insert(min1, smallest)
# If necessary, adjust the second index
if min1 <= min2:
    min2 += 1
# Return the two indices
return (min1, min2)
```

Designing Algorithms

2. STF

```
# sort_then_find3.py
whalecounts = [809, 834, 477, 478, 307, 122, 96, 102, 324, 476]
def find_two_smallest(L):
    """ (see before) """
    # Get a sorted copy of the list so that the two smallest items are at the
    # front
    temp_list = sorted(L)
    smallest = temp_list[0]
    next_smallest = temp_list[1]

    # Find their indices in the original list L
    min1 = L.index(smallest)
    min2 = L.index(next_smallest)
    # Return the indices of the two
    return (min1, min2)
```

Designing Algorithms

3. WTV

```
# walk_through7.py
whalecounts = [809, 834, 477, 478, 307, 122, 96, 102, 324, 476]
def find_two_smallest(L):
    # Examine each value in the list in order
    # Keep track of the indices of the two smallest values found so far
    # Update these values when a new smaller value is found
    # Return the two indices

...or
    # Set the first two values to smallest and smaller
    # Examine each value in the rest of the list
    # Update the minimum values when a new smaller value is found
    # Return the two indices
```

Designing Algorithms

3. WTV

```
# walk_through7.py
def find_two_smallest(L):
    """ (see before) """

    # Set min1 and min2 to the indices of the smallest and next-smallest
    # Values at the beginning of L
    if L[0] < L[1]:
        min1, min2 = 0, 1
    else:
        min1, min2 = 1, 0

    # Examine each value in the list in order
    # Update min1 and/or min2 when a new smaller value is found **
    for i in range(2, len(L)):
        if L[i] < L[min1]:
            min2 = min1
            min1 = i

        # New second smallest?
        elif L[i] < L[min2]:
            min2 = i

    # Return the two indices
    return (min1, min2)
```

Designing Algorithms

Each works. How to judge?

Profiling: speed and memory use

The larger the list the longer it takes to run, and *faster* is better

or: run multiple times

We will use the time module

```
import time
```

Designing Algorithms

```
help(time)
Help on built-in module time:

NAME
    time - This module provides various functions to manipulate time values.

DESCRIPTION
    There are two standard representations of time.  One is the number of seconds
since the Epoch, in UTC (a.k.a. GMT).  It may be an integer or a floating point
number (to represent fractions of seconds). The Epoch is system-defined; on Unix, it
is generally January 1st, 1970.
    The actual value can be retrieved by calling gmtime(0) (when time = 0)

    The other representation is a tuple of 9 integers giving local time.
    The tuple items are:
      year (including century, e.g. 1998)
      month (1-12)
      day (1-31)
      hours (0-23)
      minutes (0-59)
      seconds (0-59)
      weekday (0-6, Monday is 0)
      Julian day (day in the year, 1-366)
      DST (Daylight Savings Time) flag (-1, 0 or 1)
    If the DST flag is 0, the time is given in the regular time zone;
    if it is 1, the time is given in the DST time zone;
    if it is -1, mktime() should guess based on the date and time.
```

Designing Algorithms

```
    Variables:

    timezone -- difference in seconds between UTC and local standard time
    altzone -- difference in  seconds between UTC and local DST time
    daylight -- whether local time should reflect DST
    tzname -- tuple of (standard time zone name, DST time zone name)

    Functions:

    time() -- return current time in seconds since the Epoch as a float
    clock() -- return CPU time since process start as a float
    sleep() -- delay for a number of seconds given as a float
    gmtime() -- convert seconds since Epoch to UTC tuple
    localtime() -- convert seconds since Epoch to local time tuple
    asctime() -- convert time tuple to string
    ctime() -- convert time in seconds to string
    mktime() -- convert local time tuple to seconds since Epoch
    strftime() -- convert time tuple to string according to format specification
    strptime() -- parse string to time tuple according to format specification
    tzset() -- change the local timezone

# currect time in seconds since the Epoch
time.time()
1509388373.3531194
```

Designing Algorithms

Telling time

```
>>> print (time.strftime("%I:%M:%S")) # 12 hour
>>> print (time.strftime("%H:%M:%S")) # 24 hour
>>> print (time.strftime("%m/%d/%Y"))
>>> print (time.strftime("%d.%m.%Y"))
>>> print (time.strftime("%Y-%m-%d"))
```

Timing things

```
>>> t1 = time.perf_counter()
# Code to time goes here
>>> t2 = time.perf_counter()
>>> print('The code took {:.2f}ms'.format((t2 - t1) * 1000.))
```

We will use a program called Program_Times.py to run the three programs and time them, then tell us results.

We will work with the Darwin.slp.txt file, with 1401 lines of numbers (on Blackboard)

Designing Algorithms

```
# Program_Times.py
import time
import find_remove_find5
import sort_then_find3
import walk_through7
def time_find_two_smallest(find_func, lst):
    """ (function, list) -> float
    Return how many seconds find_func(lst) took to execute.
    """
    t1 = time.perf_counter()
    find_func(lst)
    t2 = time.perf_counter()
    return (t2 - t1) * 1000.

if __name__ == '__main__':
    # Gather the sea level pressures
    sea_levels = []
    sea_levels_file = open('darwin.slp.txt', 'r')
    for line in sea_levels_file:
        sea_levels.append(float(line))

    # Time each of the approaches
    find_remove_find_time = time_find_two_smallest(       # FRF
        find_remove_find5.find_two_smallest, sea_levels)

    sort_get_minimums_time = time_find_two_smallest(      # STF
        sort_then_find3.find_two_smallest, sea_levels)

    walk_through_time = time_find_two_smallest(           # WTV
        walk_through7.find_two_smallest, sea_levels)

    print('"Find, remove, find" took {:.2f}ms.'.format(find_remove_find_time))
    print('"Sort, get minimums" took {:.2f}ms.'.format(sort_get_minimums_time))
    print('"Walk through the list" took {:.2f}ms.'.format(walk_through_time))
```

Homework 13

Gries 12.4, page 234

Problems

1, (use function design recipe, and use sequence: 'ACTCGCTTCGCTATAAGCTAGGCAT')

2, and 6

(use:

['green','red','blue','green','red','green','blue','green','blue','green','blue' ,'red','blue','red','green','green','green','green']

)