Psychoinformatics & Neuroinformatics

Week 0

Course Introduction

by Tsung-Ren (Tren) Huang 黃從仁

Course Instructor

Tsung-Ren (Tren) Huang 黃從仁, Associate Professor

Department of Psychology

Neurobiology & Cognitive Science Center

Graduate Institute of Brain and Mind Sciences

Imaging Center for Integrated Body, Mind, & Culture Research

Institute of Applied Mathematical Sciences

Master Program in Statistics

Center for Research in Econometric Theory and Applications

Center for Artificial Intelligence and Advanced Robotics, NTU

First things first

All are welcome!?

ntu_info@googlegroups.com

李彥廷

沈家齊

Contact TAs (in person or by email) to be enrolled into NTU Cool

I "may" have recordings of this course.

Definitions of Psychoinformatics

Psychoinformatics is a relatively new interdisciplinary field that merges principles from psychology and informatics. The term "informatics" generally refers to the science of processing data for storage and retrieval. In the context of psychoinformatics, it refers to the application of computational and informational techniques to the study and understanding of psychological phenomena.

In simpler terms, psychoinformatics seeks to use computational methods, algorithms, data analysis techniques, and tools from informatics to study psychological data and derive insights about human cognition, behavior, emotions, and more.

Some common applications of psychoinformatics include:

1. Analyzing large-scale data from social media platforms to study patterns of human behavior or emotional states.
2. Developing machine learning models that can predict or understand certain psychological conditions based on digital traces.
3. Using advanced data visualization techniques to represent complex psychological phenomena.

It's an exciting field because it marries the vast and intricate world of human psychology with the powerful analytical capabilities of modern computing. The hope is that by doing so, researchers can glean new insights into the human mind and behavior that were previously inaccessible.

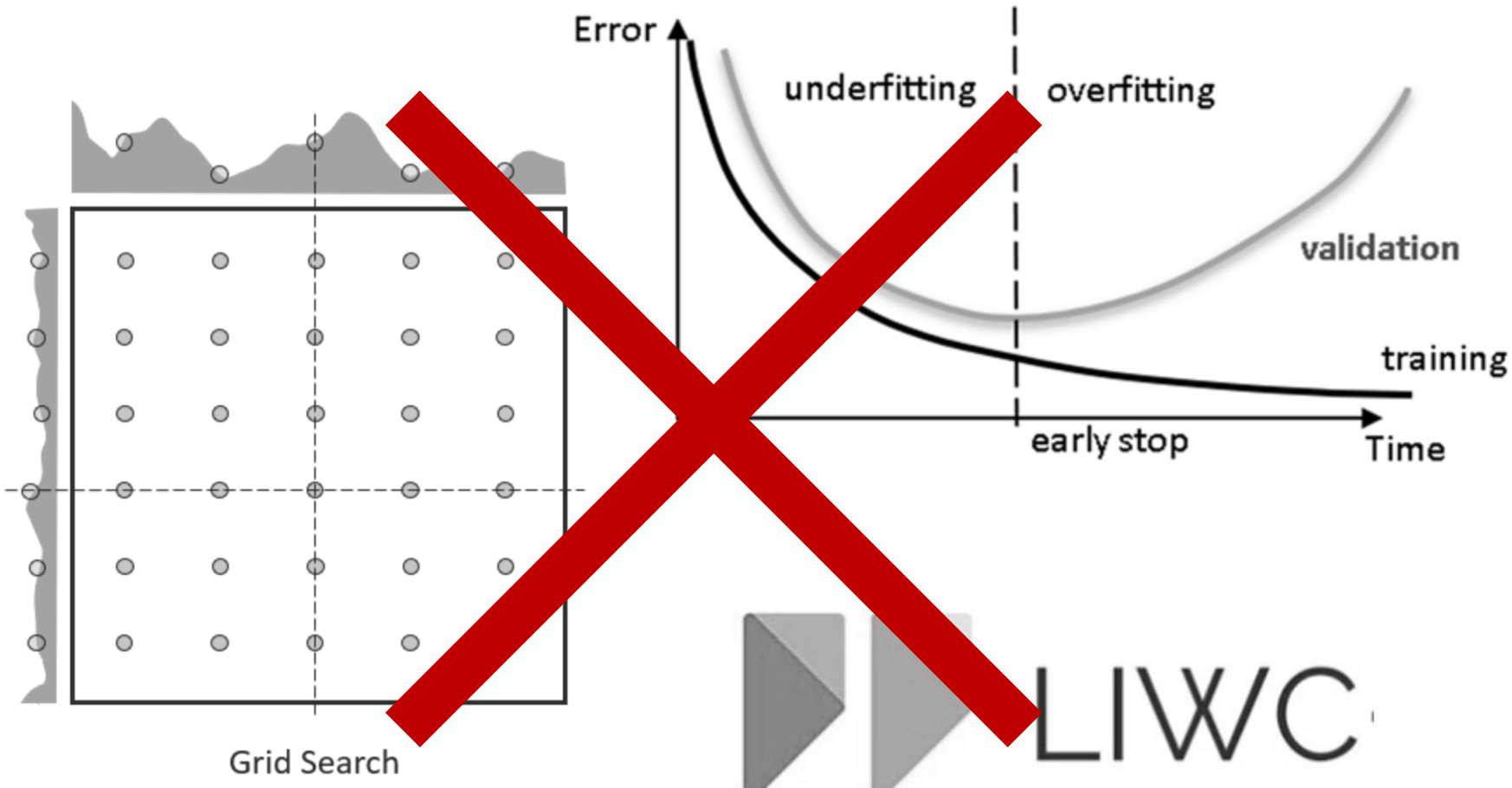
Skills needed for Psychoinformatics

Conducting psychoinformatics research requires a combination of skills from both psychology and informatics. Here are some of the core competencies and skills that would be beneficial for someone in this field:

1. **Psychological Knowledge**: A fundamental understanding of psychological theories, practices, and research methodologies is crucial. This allows the researcher to frame questions appropriately and interpret findings in a meaningful way.
2. **Research Design**: Understand how to design experiments, surveys, and observational studies. Knowledge about both quantitative and qualitative research methods can be beneficial.
3. **Programming**: Ability to code, preferably in languages popular in data science and analysis such as Python, R, or Julia. This is important for data collection, preprocessing, and analysis.
4. **Data Analysis**: Familiarity with statistical tools and techniques is essential. This includes knowledge of statistical software, understanding of machine learning algorithms, and proficiency in using data visualization tools.
5. **Big Data Handling**: Given the vast amounts of data often involved in psychoinformatics, skills in databases, data warehousing, and tools for handling big data (e.g., Hadoop, Spark) can be beneficial.
6. **Natural Language Processing (NLP)**: As many psychoinformatics studies may involve analyzing textual data from sources like social media, familiarity with NLP techniques can be useful.
7. **Ethical Considerations**: Understanding the ethical implications of data collection, especially from online sources, and ensuring privacy and confidentiality of data subjects.
8. **Soft Skills**: Like any research field, good communication skills, both written and verbal, are crucial for publishing findings, collaborating with other researchers, and conveying insights to non-experts.
9. **Domain-specific Tools**: Familiarity with tools or platforms specific to the area of study (e.g., software for neuroimaging analysis if the research involves brain imaging).
10. **Collaboration and Interdisciplinary Understanding**: The interdisciplinary nature of psychoinformatics means that collaboration is often necessary. Being able to work with experts from different backgrounds and understanding their terminologies and methods is crucial.
11. **Continuous Learning**: Given the rapidly evolving nature of technology and the increasing intersections of psychology with new areas of informatics, a mindset geared towards continuous learning and adaptability is beneficial.

It's worth noting that while having a wide range of skills is beneficial, many researchers in psychoinformatics might specialize in certain areas and collaborate with others who complement their skill sets. For example, a researcher might be highly skilled in psychological theory and experimental design but collaborate with a data scientist for the computational analysis portion of the research.

Continuous Learning



LIWC

Bard

My Teaching Goals

Expose you to the forefront of psychoinformatics/neuroinformatics

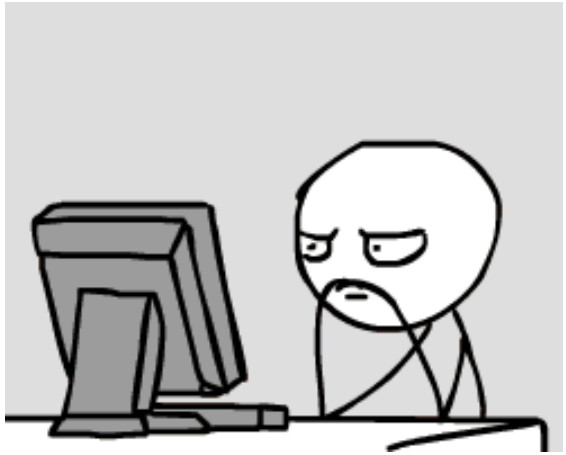
Let's stay current with the latest research

Help you figure out whether this direction is for you

If not, you have also further narrowed your career search

Help you become a life-long learner & performer

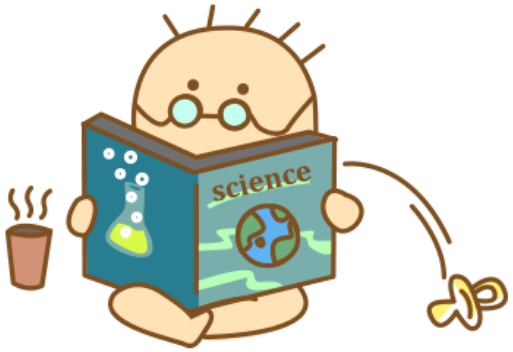
Primarily self-learning by doing & debugging



Scientifically, what this course offers

Tools that allow for the pursuit of ecological validity

Ecological validity is a major trend in human research.



Understanding of the pros/cons about big-data *collection* methods

Big data are often not fine data.

Understanding of the pros/cons about big-data *analysis* methods

Results from deep learning may be irreproducible or misleading.

Practically, what this course offers

Tools for observing/predicting real-life human behavior

High demands in human-related businesses

Preparation for a career in Data Engineering

Learning techniques about data *collection*

Preparation for a career in Data Science

Learning techniques about data *analysis*

Exactly, what this course offers

週次	日期	單元主題
第1週	9/8	課程簡介+基本程式設計 (Python)+基本資料分析 (NumPy & Pandas)
第2週	9/15	單機版實驗程式的設計 (PsychoPy & Socket Programming)
第3週	9/22	網路資料的搜集1/2 (Web APIs)
第4週	9/29	中秋節(放假)
第5週	10/6	網路資料的搜集2/2 (LXML, Scrapy, & Selenium)
第6週	10/13	網頁與手機實驗1/3 (Frontend: Javascript)
第7週	10/20	網頁與手機實驗2/3 (Backend & Databases: Node.js, FastAPI, & SQLite)
第8週	10/27	網頁與手機實驗3/3 (Smartphone Apps: PWA, Hybrid Apps, Compiled Apps)
第9週	11/3	機器學習的應用1/3 (Scikit-learn: Unsupervised & Supervised Learning; Causal ML)
第10週	11/10	機器學習的應用2/3 (Advanced topics: Hyperparameter tuning & Ensemble models)
第11週	11/17	機器學習的應用3/3 (Deep Learning: Keras; XAI)
第12週	11/24	文字資料的處理 (Regular Expressions & Basic NLP)
第13週	12/1	影像資料的處理 (Image Processing & Computer Vision)
第14週	12/8	聲音資料的處理 (Audio & Speech Processing; Chatbots)
第15週	12/15	巨量資料的處理 (Asynchronous, Parallel, & Distributed Computing)
第16週	12/22	期末考(不上課)

Psychoinformatics & Neuroinformatics

Week 0.5

Python Basics

by Tsung-Ren (Tren) Huang 黃從仁

Programming Languages

Python Environments

Personal Computing:

Recommend VS Code + Extensions (Python + Jupyter)

Private Cloud:

Recommend Jupyter Notebook in Anaconda Python 3

Public Cloud:

Thanks Google for giving $50 credits/student!

Recommend Google Colab, especially for AI & deep learning

Copilot with ChatGPT

VS Code:

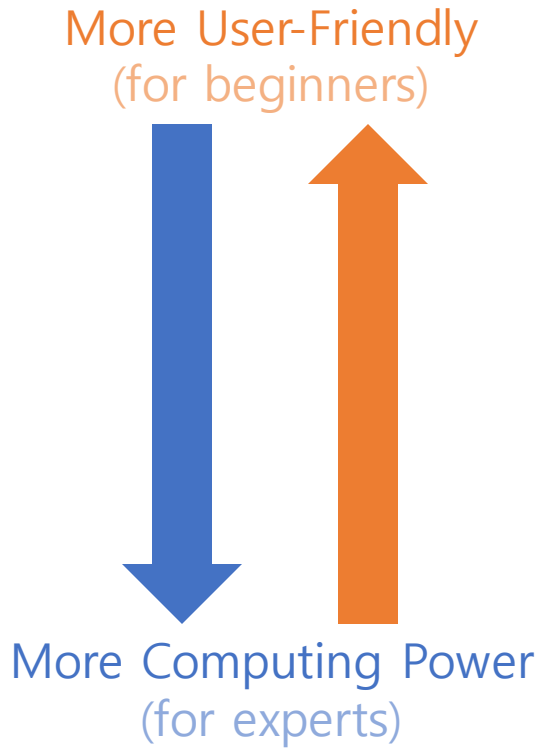
Install these extensions: GitHub Copilot + ChatGPT

Jyputer:

Install this Chrome/Edge/Brave extension

Colab:

Install this Chrome/Edge/Brave extension



VS Code/Jupyter/Colab

Use [Shift] + [Enter] for execution

Remember this at least

Use [Tab] for auto-completion (not for Colab)

Or for exploration of functions under a library

Use [Shift] + [Tab] for help (not for VS Code & Colab)

Holding [Shift] + more [Tab] to get more info

PYTHON FOR DATA SCIENCE CHEAT SHEET

Python Basics

Datatypes

- Numbers: a=2(Integer), b=2.0(Float), c=1+2j(Complex)
- List: a=[1,2,3,'Word']
- Tuple: a= (1,2,4)
- String: a="New String"
- Sets: a= {2,3,4,5}
- Dictionary: x= {'a': [1,2],'b': [4,6]}

Operators

Numeric Operator: (Say, a holds 5, b holds 10)

- a + b = 15
- a – b = -5
- a * b = 50
- 7.0//2.0 = 3.0, -11//3 = -4
- b/a = 2
- b % a = 0
- a**b =9765625

Comparison Operator:

- (a == b): not true
- (a!= b): true
- (a > b): not true
- (a > b): not true
- (a >= b): not true
- (a <= b) is true

Boolean Operator:

- a and b
- a or b
- not a

Operations

List Operations

- List=[]: Defines an empty list
- list[i]=a: Stores a at the ith position
- list[i]: Retrieves the character at the ith position
- list[i:j]: Retrieves characters in the range i to j
- list.append(val): Adds item at the end
- list.pop([i]): Removes and returns item at index i

String Operations

- String[i]: Retrieves the character at the ith position
- String[i:j]: Retrieves characters in the range i to j

Dictionary Operations

- dict={} : Defines an empty dictionary
- dict[i]=a: stores "a" to the key "i"
- dict[i]: Retrieves the item with the key "i"
- dict.key: Gives all the key items
- dict.values: Gives all the values

OOPS

Inheritance:

A process of using details from a new class without modifying existing class.

Polymorphism:

A concept of using common operation in different ways for different data input.

Encapsulation:

Hiding the private details of a class from other objects.

Class/object

Class:

```
class Pen:
    pass
```

Object: obj=Pen()

Flow Control Method

- **if-else** (Conditional Statement)

```
if price>=700:
        print("Buy.")
else:
        print("Don't buy.")
```

- **For loop** (Iterative Loop Statement)

```
a="New Text"
count=0
for i in a:
    if i=='e':
        count=count+1
print(count)
```

- **While loop** (Conditional Loop Statement)

```
a=0
i=1
while i <10:
        a=a*2
        i=i+1
print(a)
```

- **Loop Control:** Break, Pass and continue

Functions

```
def new_function():
    print("Hello World")

new_function()
```

Lambda Function

```
lambda a,b: a+b

lambda a,b: a*b
```

Comments

```
# Single Line Comment
"""
Multi-line comment
"""
```

Generic Operations

- range(5): 0,1,2,3,4
- S=input("Enter:")
- Len(a): Gives item count in a
- min(a): Gives minimum value in a
- max(a): Gives minimum value in a
- sum(a): Adds up items of an iterable and returns sum
- sorted(a): Sorted list copy of a
- importing modules: import random

File Operations

f= open("File Name","opening mode")

(Opening modes: r: read, w: write, a: append, r+: both read and write)

Try & Except Block

```
try:
        [Statement body block]
        raise Exception()
except Exception as e:
        [Error processing block]
```

FURTHERMORE:

Python for Data Science Certification Training Course

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

```
>>> x+2
 7
>>> x-2
 3
>>> x*2
 10
>>> x**2
 25
>>> x%2
 1
>>> x/float(2)
 2.5
```

Sum of two variables
Subtraction of two variables
Multiplication of two variables
Exponentiation of a variable
Remainder of a variable
Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset

```
>>> my_list[1]
>>> my_list[-3]
```

Select item at index 1
Select 3rd last item

Slice

```
>>> my_list[1:3]
>>> my_list[1:]
>>> my_list[:3]
>>> my_list[:]
```

Select items at index 1 and 2
Select items after index 0
Select items before index 3
Copy my_list

Subset Lists of Lists

```
>>> my_list2[1][0]
>>> my_list2[1][:2]
```

my_list[list][itemOfList]

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

```
>>> my_list.index(a)
>>> my_list.count(a)
>>> my_list.append('!')
>>> my_list.remove('!')
>>> del(my_list[0:1])
>>> my_list.reverse()
>>> my_list.extend('!')
>>> my_list.pop(-1)
>>> my_list.insert(0,'!')
>>> my_list.sort()
```

Get the index of an item
Count an item
Append an item at a time
Remove an item
Remove an item
Reverse the list
Append an item
Remove an item
Insert an item
Sort the list

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper()
>>> my_string.lower()
>>> my_string.count('w')
>>> my_string.replace('e', 'i')
>>> my_string.strip()
```

String to uppercase
String to lowercase
Count String elements
Replace String elements
Strip whitespaces

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

Data analysis
Machine learning
Scientific computing
2D plotting

Install Python

ANACONDA
Leading open data science platform powered by Python

spyder
Free IDE that is included with Anaconda

jupyter
Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset

```
>>> my_array[1]
  2
```

Select item at index 1

Slice

```
>>> my_array[0:2]
  array([1, 2])
```

Select items at index 0 and 1

Subset 2D Numpy arrays

```
>>> my_2darray[:,0]
  array([1, 4])
```

my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
 array([False, False, False,  True], dtype=bool)
>>> my_array * 2
 array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

Numpy Array Functions

```
>>> my_array.shape
>>> np.append(other_array)
>>> np.insert(my_array, 1, 5)
>>> np.delete(my_array,[1])
>>> np.mean(my_array)
>>> np.median(my_array)
>>> my_array.corrcoef()
>>> np.std(my_array)
```

Get the dimensions of the array
Append items to an array
Insert items in an array
Delete items in an array
Mean of the array
Median of the array
Correlation coefficient
Standard deviation

CHATGPT

You

Creaet a tutorial on Python basics with codes

ChatGPT

Certainly! Here's a tutorial on Python basics that covers fundamental concepts along with code examples:

Installing Python:

Visit the official Python website (python.org) and download the latest version of Python.

Follow the installation instructions for your operating system.

Running Python Code:

Open a text editor or an Integrated Development Environment (IDE) like PyCharm, Visual Studio Code, or IDLE.

Write your Python code in a new file with a .py extension.

Save the file.

Open the terminal or command prompt, navigate to the directory where your code file is saved, and run the command `python filename.py` to execute the code.

Print Statements:

Print statements are used to display output in Python.

Use the `print()` function to print text, variables, or expressions.

Example:

```python
print("Hello, World!")
```

Variables and Data Types:

Variables are used to store data.

Python supports various data types such as integers, floating-point numbers, strings, booleans, lists, tuples, and dictionaries.

Example:

```python
age = 25
name = "John"
is_student = True
```

User Input:

Use the `input()` function to prompt the user for input.

The `input()` function returns a string, so you may need to convert it to the appropriate data type.

Example:

```python
```

Enter to send, Shift+Enter for a new line, / to choose prompt

test.ipynb

test.ipynb > a=1

+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Outline ··· Python 3.9.6

```
print("Hello, World!")
```

[2] ✓ 0.0s Python

··· 4

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER Tasks

Ln 1, Col 23 Spaces: 4 LF

More on “import”

To be clear:

Import random; random.random()

To be friend:

Import random as rnd; rnd.random()

To be lazy:

from random import *; random()

import this
import antigravity

More on “if”: A/B Testing


```
import random
r=random.random()
if(r>0.5):
        print("Version A")
else:
        print("Version B")
```

More on “for”

Iterating through each element/case:

```
for i in [5566,'never dies',range(3)]:
        print(i)
```

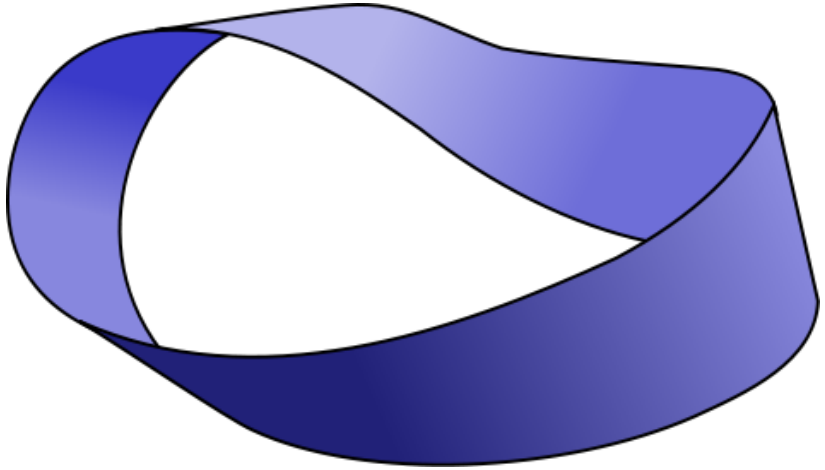
Repeating operations:

```
for i in range(3):
        print("I don't care i")
```

More on “while”

Number of repetitions is unknown:

```
import random
c=0 #counter
acc=0 #participant accuracy
while(acc<0.9):
        c=c+1
        acc=random.random() #simulated performance
        print(c,r)
```



Put together: Reaction Time

```
import time,random
print("Get ready...",flush=True)
time.sleep(10*random.random())
t0=time.time()
input("Press [Enter] now!")
print(time.time()-t0)
```

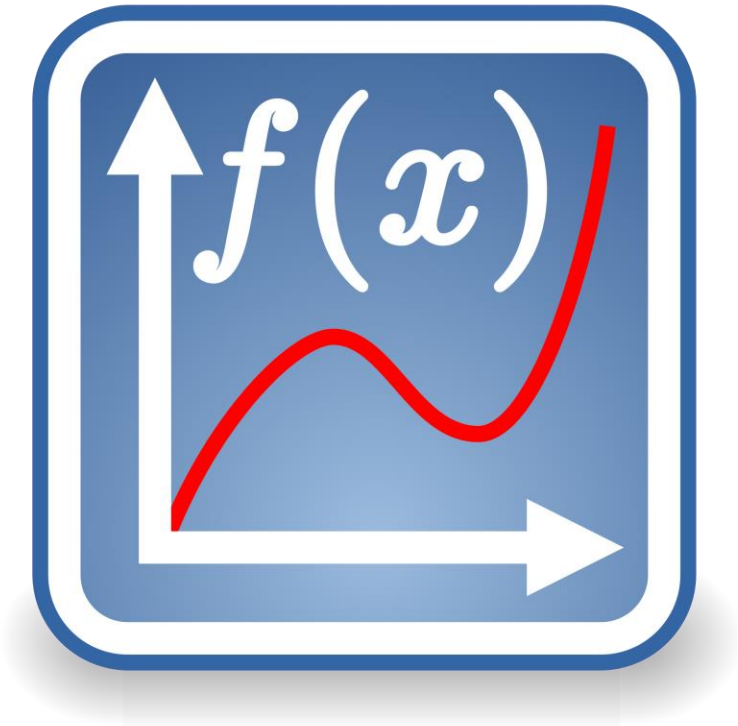
Psychoinformatics & Neuroinformatics

Week 1

Basic Data Analysis

by Tsung-Ren (Tren) Huang 黃從仁

The story begins with this function…



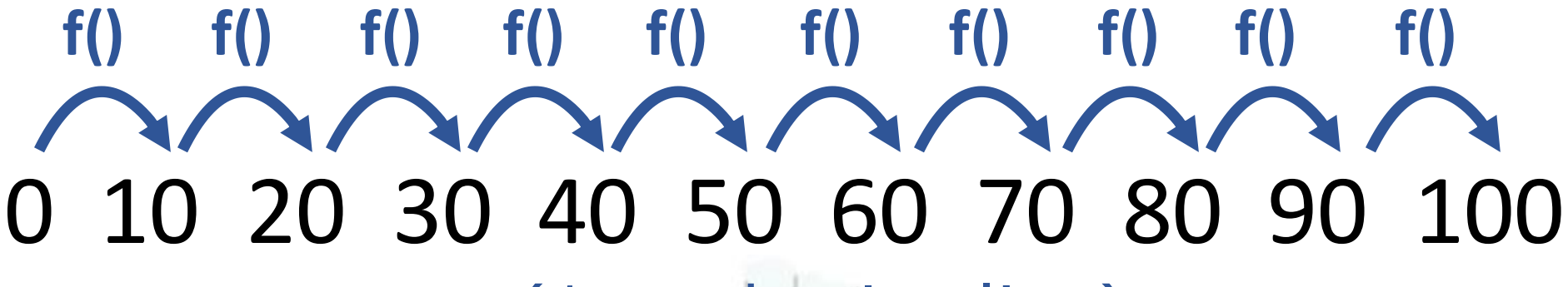

```
import math
def adjust_score(old):
    new=math.sqrt(old)*10
    return new

a=adjust_score(0)
b=adjust_score(60)
print(a,b)
```

Single Instruction, Multiple Data (SIMD)

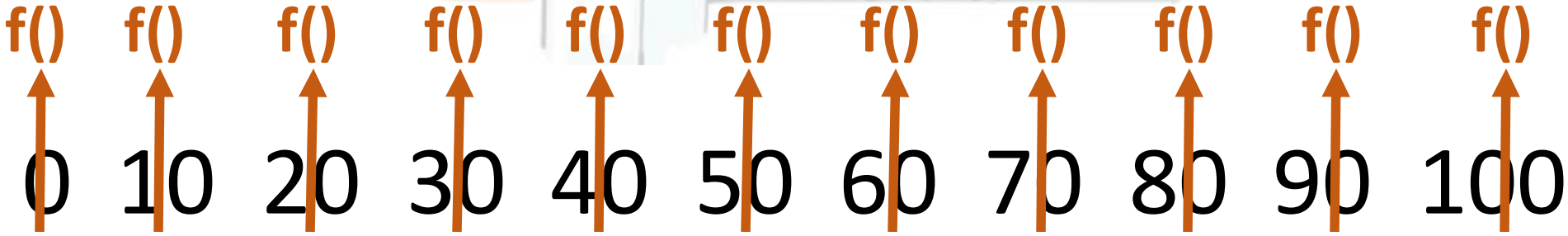
```
print(adjust_score(range(0,101,10)))
```

TypeError: a float is required

Sequential Computing vs. Parallel Computing



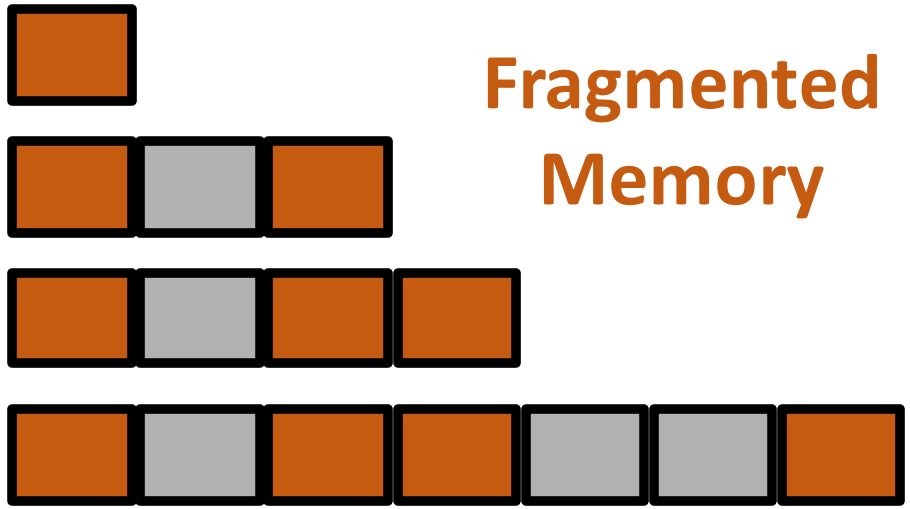
(1 production line)



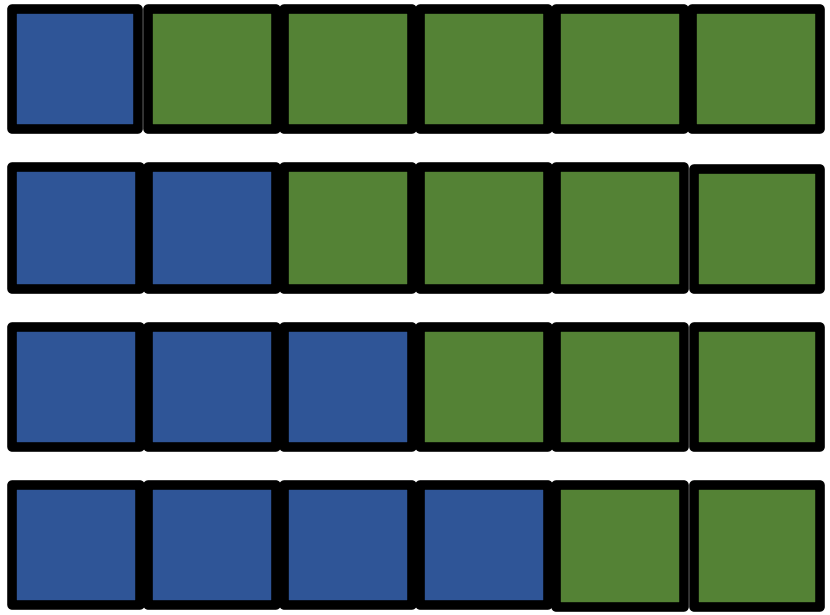
(11 production lines)

Solutions 1: Sequential

```
scores=[]
for i in range(0,101,10):
        scores.append(adjust_score(i))
print(scores)
```




```
old=range(0,101,10)
N=len(old)
scores=[0.]*N
for i in range(N):
        scores[i]=adjust_score(i)
print(scores)
```



Solution 2: Seemingly Parallel

```
import math
def adjust_score(old):
        new=math.sqrt(old)*10
        return new

print(list(map(adjust_score,range(0,101,10))))
```

Unlike the map() in multiprocessing or MapReduce, the map() here is actually a sequential operation.

Solution 3: The NumPy way!

```
import numpy as np; (np.arange(0,101,10)**0.5)*10 #Just one line!
```

NumPy's *a*range() vs. built-in range()

```
a=range(0,101,10); b=np.arange(0,101,10)
a+1
b+1
a+a
b+b
a*3
b*3
```

⇦ **Vectorized operation = Element-wise operation**
(c,d)+(e,f)=(c+e,d+f)

Another Example: 100 random numbers

Use random.random() for 100 times:

```
import random
r=[]
for i in range(100):
        r.append(random.random())
```


Use numpy.random.rand() for 1 time:

```
import numpy as np
r=np.random.rand(100)
```

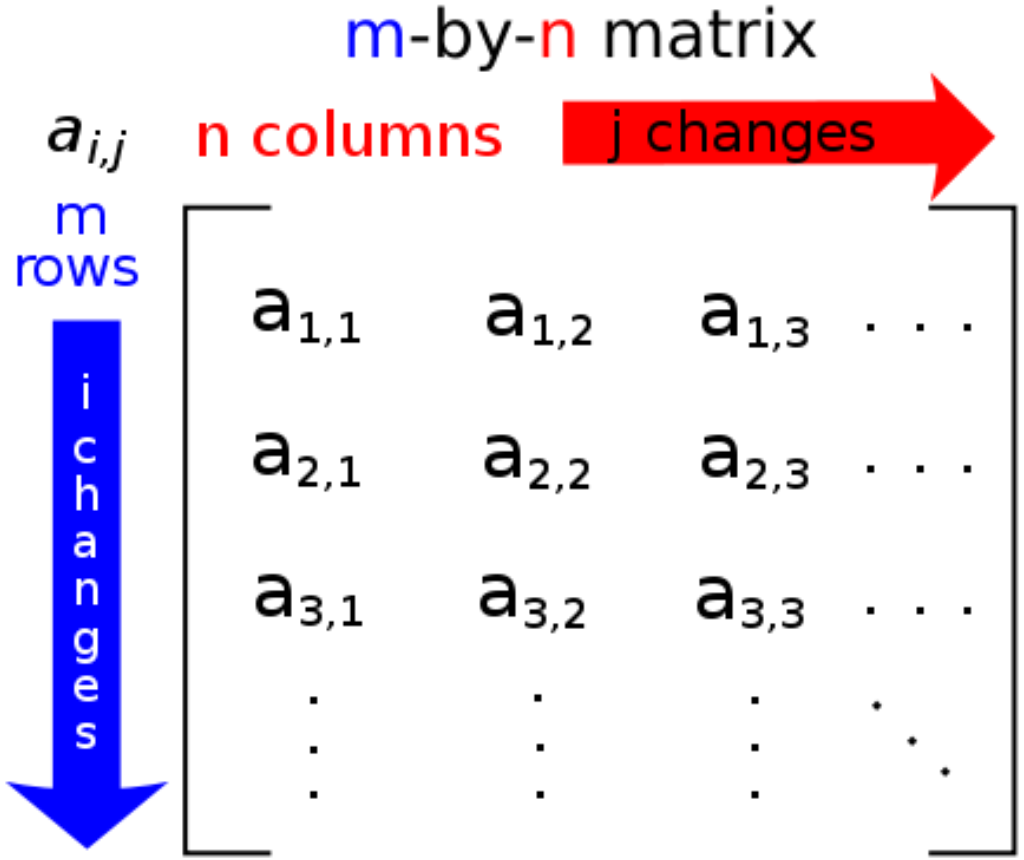
List vs. NumPy Array: *Irregular* List

```
a=[[5566,'never dies'],5,[[['R',range(3)],'doll'],6]]
a[0] #[5566, 'never dies']
a[1] #5
a[0][0] #5566
a[2][0][0][1] #range(0,3)
```

Tip: Think of a multi-level list as a tree

List vs. NumPy Array: *Regular* Array

```
a=np.array([range(3),np.random.rand(3)])
a.dtype #dtype('float64')
a.T # transpose
a[0][2] #2.0
a[0,2] #2.0
a[0,:] #array([ 0.,  1.,  2.])
a[0,1:3] #array([ 1.,  2.])
np.mean(a)
np.mean(a,0)
np.mean(a,1)
```



datacamp

Python For Data Science
NumPy Cheat Sheet

Learn NumPy online at www.DataCamp.com

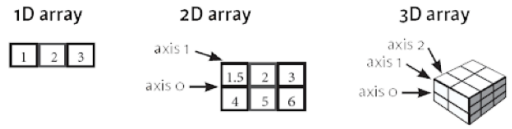
Numpy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



> Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)],[(3,2,1), (4,5,6)]], dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4)) #Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16) #Create an array of ones
>>> d = np.arange(10,25,5) #Create an array of evenly spaced values (step value)
>>> np.linspace(0,2,9) #Create an array of evenly spaced values (number of samples)
>>> e = np.full((2,2),7) #Create a constant array
>>> f = np.eye(2) #Create a 2X2 identity matrix
>>> np.random.random((2,2)) #Create an array with random values
>>> np.empty((3,2)) #Create an empty array
```

> I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

> Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

> Inspecting Your Array

```
>>> a.shape #Array dimensions
>>> len(a) #Length of array
>>> b.ndim #Number of array dimensions
>>> e.size #Number of array elements
>>> b.dtype #Data type of array elements
>>> b.dtype.name #Name of data type
>>> b.astype(int) #Convert an array to a different type
```

> Data Types

```
>>> np.int64 #Signed 64-bit integer types
>>> np.float32 #Standard double-precision floating point
>>> np.complex #Complex numbers represented by 128 floats
>>> np.bool #Boolean type storing TRUE and FALSE values
>>> np.object #Python object type
>>> np.string_ #Fixed-length string type
>>> np.unicode_ #Fixed-length unicode type
```

> Array Mathematics

Arithmetic Operations

```
>>> g = a - b #Subtraction
 array([[-0.5, 0. , 0. ],
        [-3. , -3. , -3. ]])
>>> np.subtract(a,b) #Subtraction
>>> b + a #Addition
 array([[ 2.5, 4. , 6. ],
        [ 5. , 7. , 9. ]])
>>> np.add(b,a) #Addition
>>> a / b #Division
 array([[ 0.66666667, 1. , 1. ],
        [ 0.25 , 0.4 , 0.5 ]])
>>> np.divide(a,b) #Division
>>> a * b #Multiplication
 array([[ 1.5, 4. , 9. ],
        [ 4. , 10. , 18. ]])
>>> np.multiply(a,b) #Multiplication
>>> np.exp(b) #Exponentiation
>>> np.sqrt(b) #Square root
>>> np.sin(a) #Print sines of an array
>>> np.cos(b) #Element-wise cosine
>>> np.log(a) #Element-wise natural logarithm
>>> e.dot(f) #Dot product
 array([[ 7., 7.],
        [ 7., 7.]])
```

Comparison

```
>>> a == b #Element-wise comparison
 array([[False, True, True],
        [False, False, False]], dtype=bool)
>>> a < 2 #Element-wise comparison
 array([True, False, False], dtype=bool)
>>> np.array_equal(a, b) #Array-wise comparison
```

Aggregate Functions

```
>>> a.sum() #Array-wise sum
>>> a.min() #Array-wise minimum value
>>> b.max(axis=0) #Maximum value of an array row
>>> b.cumsum(axis=1) #Cumulative sum of the elements
>>> a.mean() #Mean
>>> b.median() #Median
>>> a.corrcoef() #Correlation coefficient
>>> np.std(b) #Standard deviation
```

> Copying Arrays

```
>>> h = a.view() #Create a view of the array with the same data
>>> np.copy(a) #Create a copy of the array
>>> h = a.copy() #Create a deep copy of the array
```

> Sorting Arrays

```
>>> a.sort() #Sort an array
>>> c.sort(axis=0) #Sort the elements of an array's axis
```

> Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2] #Select the element at the 2nd index
 3
>>> b[1,2] #Select the element at row 1 column 2 (equivalent to b[1][2])
 6.0
```

Slicing

```
>>> a[0:2] #Select items at index 0 and 1
 array([1, 2])
>>> b[0:2,1] #Select items at rows 0 and 1 in column 1
 array([ 2., 5.])
>>> b[:1] #Select all items at row 0 (equivalent to b[0:1, :])
 array([[1.5, 2., 3.]])
>>> c[1,...] #Same as [1,:,:]
 array([[[ 3., 2., 1.],
         [ 4., 5., 6.]]])
>>> a[ : :-1] #Reversed array a array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2] #Select elements from a less than 2
 array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]] #Select elements (1,0),(0,1),(1,2) and (0,0)
 array([ 4. , 2. , 6. , 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]] #Select a subset of the matrix's rows and columns
 array([[ 4. ,5. , 6. , 4. ],
        [ 1.5, 2. , 3. , 1.5],
        [ 4. , 5. , 6. , 4. ],
        [ 1.5, 2. , 3. , 1.5]])
```

> Array Manipulation

Transposing Array

```
>>> i = np.transpose(b) #Permute array dimensions
>>> i.T #Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel() #Flatten the array
>>> g.reshape(3,-2) #Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6)) #Return a new array with shape (2,6)
>>> np.append(h,g) #Append items to an array
>>> np.insert(a, 1, 5) #Insert items in an array
>>> np.delete(a,[1]) #Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0) #Concatenate arrays
 array([ 1, 2, 3, 10, 15, 20])
>>> np.vstack((a,b)) #Stack arrays vertically (row-wise)
 array([[ 1. , 2. , 3. ],
        [ 1.5, 2. , 3. ],
        [ 4. , 5. , 6. ]])
>>> np.r_[e,f] #Stack arrays vertically (row-wise)
>>> np.hstack((e,f)) #Stack arrays horizontally (column-wise)
 array([[ 7., 7., 1., 0.],
        [ 7., 7., 0., 1.]])
>>> np.column_stack((a,d)) #Create stacked column-wise arrays
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
>>> np.c_[a,d] #Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3) #Split the array horizontally at the 3rd index
 [array([1]),array([2]),array([3])]
>>> np.vsplit(c,2) #Split the array vertically at the 2nd index
 [array([[[ 1.5, 2. , 1. ],
          [ 4. , 5. , 6. ]]]),
 array([[[ 3., 2., 3.],
         [ 4., 5., 6.]]])]
```

NumPy Application: Experimental Design

Let's generate 15 trials for 3 experimental conditions

Ideally an equal number of trials per condition

Randomized design:

```
import numpy as np; trials=np.random.randint(0,3,15)
```

Counterbalanced design:

```
import numpy as np;
trials=np.array(list(range(3))*5) #Exactly 5 trials per condition
trials=np.random.permutation(trials)
```

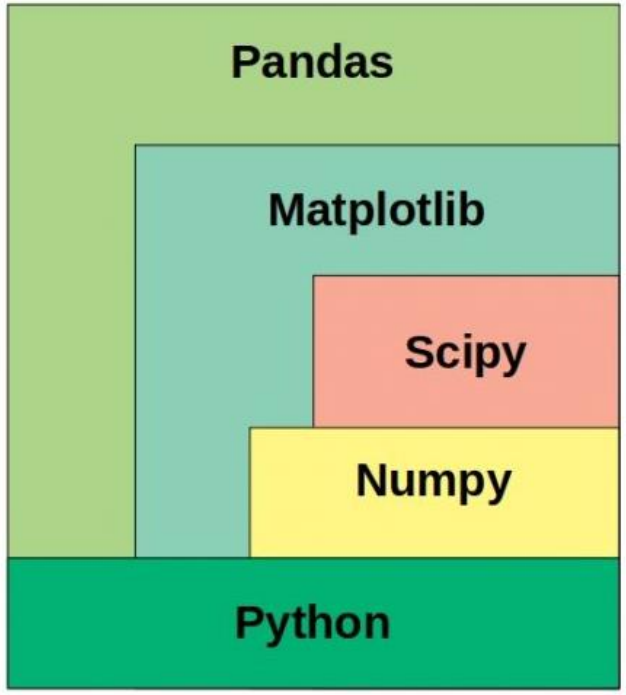
NumPy Application: Basic Data Analysis

Cond.	Score	Reaction Time
1	1	-1 (timed out)
0	0	0.444112
2	1	2.597051
…	…	…

```
import numpy as np
data=np.loadtxt('exp_subj0.txt')
valid=(data[:,2]>0) # RT>0
data=data[valid,:] # valid trial
print(np.mean(data[:,1])) #mean acc.
print(np.mean(data[:,2])) # mean RT
```

```
Ngroups=np.unique(data[:,0]).size #3
groups=[0]*Ngroups #[0, 0, 0]
for i in range(Ngroups):
        selector=(data[:,0]==i) # group 0, 1, or 2?
        groups[i]=data[selector,:] # use list to store arrays
```

NumPy vs. Pandas



Pandas	NumPy
Can contain Dissimilar Data Types	Has Homogeneous Data
Tabular operations, SQL like semantics Preprocessing tasks	Numeric computing, Matrix and vector operations
Two dimensions	Multidimensional (> 2 possible)
More Memory	Less Memory
Slower	Faster

```
import pandas as pd
df=pd.read_table('exp_subj0.txt',sep=' ')
df.describe() # ~ R's summary
```

Data Wrangling

with pandas Cheat Sheet

http://pandas.pydata.org

Pandas API Reference Pandas User Guide

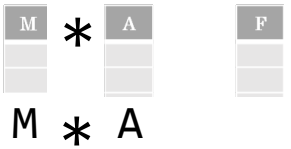
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4, 5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```
Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```
Specify values for each row.

		a	b	c
N	v			
D	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
```
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

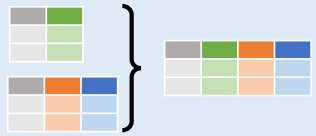
```
df = (pd.melt(df)
        .rename(columns={
                'variable':'var',
                'value':'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change layout, sorting, reindexing, renaming

pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.

pd.concat([df1,df2])
Append rows of DataFrames



pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg', ascending=False)
Order rows by values of a column (high to low).

df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length', 'Height'])
Drop columns from DataFrame

Subset Observations - rows

df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10) Randomly select n rows.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

Subset Variables - columns

df[['width', 'length', 'species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
         engine="python")
```

Subsets - rows and columns

Use **df.loc[]** and **df.iloc[]** to select only rows, only columns or both.
Use **df.at[]** and **df.iat[]** to access a single value by row and column.
First index selects rows, second index columns.

df.iloc[10:20]
Select rows 10-20.

df.iloc[:, [1, 2, 5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[:, 'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.loc[df['a'] > 10, ['a', 'c']]
Select rows meeting logical condition, and only the specific columns .

df.iat[1, 2] Access single value by index

df.at[4, 'A'] Access single value by label

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

Cheatsheet for pandas (http://pandas.pydata.org) originally written by Irv Lustig, Princeton Consultants, inspired by RStudio Data Wrangling Cheatsheet

Homework to NTU Cool by 9/15(Fri)

Use pandas to analyze the data of a study about power poses:

Game Over