videos on linear algebra

The Essence of Linear Algebra series (3blue1brown) linked below is an excellent resource for reviewing (or learning) basic concepts of linear algebra. The most important concepts to review are vectors (Chapter 1-2), linear transformations and matrices (Chapter 3), matrix multiplication (Chapter 4), non-square matrices as transformations between dimensions (Chapter 8), and dot products (Chapter 9).

Please review these.

https://www.youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab

Homework 4

posted on Brightspace

due Mon (Oct 3) at 11:59pm*

(pushed back so I'm not lecturing about topics for a homework due two days later)

(due late on Mon so people can ask last-minute questions that Mon after class)

`Homework4.pdf` (written description)

`Homework4.ipynb` (notebook to use for your solution)

`difdata.csv`

* most homeworks will continue to be due at class time

download from Brightspace

`FileIO.ipynb`
`LogicalIndexing.ipynb`

some basic file I/O

`FileIO.ipynb`

binary vs. ASCII file formats

binary vs. ASCII file formats

Every file is a "binary" file in the sense that its contents is merely bits (0s and 1s) and bytes (8 bits).

binary vs. ASCII file formats

Every file is a "binary" file in the sense that its contents is merely bits (0s and 1s) and bytes (8 bits).

e.g.,

<u>bits</u>

01001101

binary vs. ASCII file formats

Every file is a "binary" file in the sense that its contents is merely bits (0s and 1s) and bytes (8 bits).

bits

$$\underbrace{0100}_{4}\underbrace{1101}_{D}$$

binary	hex
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

binary vs. ASCII file formats

Every file is a "binary" file in the sense that its contents is merely bits (0s and 1s) and bytes (8 bit).

binary (base 2) bits	hexadecimal (base 16) bytes	ASCII character
01001101	4D	M
01101101	6D	m
00100011	23	#

ASCII "binary" files are interpreted universally (8 bit). Unicode is a 16 or 32 bit extension (non-western characters, emojis, other) - ASCII is a subset of Unicode.

https://www.youtube.com/watch?v=1SMmc9gQmHQ

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII refers to a symbols. Keycodes refer to a keyboard key.

binary vs. ASCII file formats

ASCII files are read by every operating system and many many programs exactly the same way.

the extension (.txt) by itself doesn't make it a "text file" (ASCII), but can give a clue to the O/S about what default program to use to read the file

text files (.txt)

configuration files (.ini)

batch files (.bat)

scripts and source code (.m, .c, .py)

notebooks (.ipynb)

if you have a data file (or program code) that is in ASCII format you are guaranteeing it can be read by anyone anywhere in the world (likely "forever")

binary vs. ASCII file formats

binary files are also made of bits and bytes but the way those bits and bytes are interpreted depends on the particular O/S and program

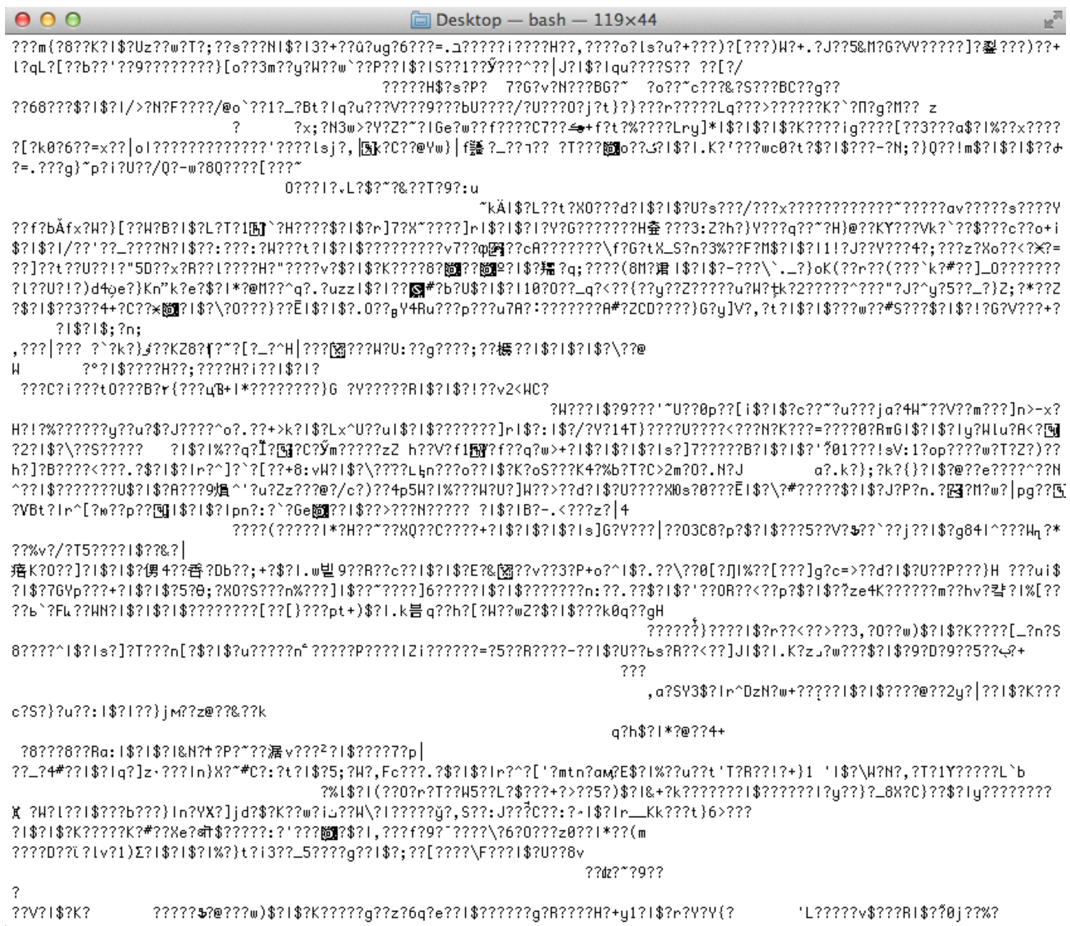
you either need to have the program (and perhaps the O/S) that created the file or hope that a newer version (or a different program) can read those files

e.g., I have some analysis files from graduate school created by a program that hasn't existed for a decade

e.g., I couldn't read Microsoft Word files created 15 years ago in Microsoft Word by Microsoft Word now

binary vs. ASCII file formats

If you open a binary file in an application that doesn't understand it, it probably defaults to an ASCII interpretation, which could be garbage ...



basic File I/O in Python

(and a bit on iterators and navigating arrays)

variety of file I/O approaches in Python

human-readable

- CSV file (ASCII text, comma-delimited)
  general-purpose

  human-readable

- JSON file (ASCII text, more complex)
  general-purpose

- `numpy.save`/`numpy.load` create/read `.npy` files (binary)
  Python-specific

- `pickle` and `marshal` (binary)
  Python-specific

- HDF5 format (binary)
  general-purpose

of course there are other binary, general-purpose file formats for images (.jpg, .gif, .tif, etc.), audio (.mp3, .wav, etc.), and video (.mp4, .mov, etc.)

ASCII text files

CSV file I/O

CSV = "Comma*-Separated Value" format

* a different delimiter can be specified

data.csv — Edited

```
TP,JA,WO,WM,KL,WS,BB,EC,VB
0.89,0.91,0.78,0.82,0.74,0.81,0.88,0.92,0.83
0.71,0.73,0.81,0.73,0.76,0.83,0.7,0.72,0.83
451,515,574,614,550,643,634,514,613
723,689,712,613,812,743,690,719,772
```

file needs to be read row by row
and parsed (following the delimiters) into the right data structures with the right types

spreadsheets (Excel, Numbers)** can read/write CSV format

many analysis programs read/write CSV format

** lose formulas and formatting when doing so

https://en.wikipedia.org/wiki/Comma-separated_values

CSV file I/O

CSV = "Comma-Separated Value" format

critically, they're also ASCII (text) file format (can be read by any program that can read text files) - raw text files, JSON, XML are other ASCII formats

Best Practices : always avoid saving important information (only) in binary (non-ASCII) format (especially data)

binary files can only be read by the applications that created them (and those applications might stop working)

e.g., I recently accessed data (text) files I collected 30 years ago

CSV file I/O

```
# subject initials
sinit = ['TP', 'JA', 'WO', 'WM', 'KL', 'WS', 'BB', 'EC', 'VB']

# subject accuracy
acc = np.array([[.89, .91, .78, .82, .74, .81, .88, .92, .83],
                [.71, .73, .81, .73, .76, .83, .70, .72, .83]])

# subject mean RTs
rts = np.array([[451, 515, 574, 614, 550, 643, 634, 514, 613],
                [723, 689, 712, 613, 812, 743, 690, 719, 772]])
```

https://docs.python.org/3/library/csv.html

CSV file I/O

can save 1-dimensional and 2-dimensional data in a CSV file (easily)

```
import csv
```

opens file
for writing

a robust way to open files in Python (prevent issues if they are not closed properly)

```
with open('sinit.csv', 'w') as fp:
    csvwriter = csv.writer(fp)
    csvwriter.writerow(sinit)
```

`fp` is the "file pointer"

creates a "writer" object to fp

writes 1D object to a row in fp

not using `with`, you would need to call `fp.close()`

```
with open('acc.csv', 'w') as fp:
    csvwriter = csv.writer(fp)
    csvwriter.writerows(acc)
```

writes 2D object to rows in fp

`with` in Python for file I/O

`with` in Python (for files) does opening and closing of files and takes care of exception handling gracefully (if file not found, or an error)

```
with open('sinit.csv', 'w') as fp:
```

stuff here

stuff here

.

.

.

files are external resources with respect to Python (`with` works with other kinds of resources) - they do a lot of things out of your control (e.g., writing is often buffered, and if you fail to properly close a file, information buffered but not written will not be added to the file) - `with` takes care of that for you

https://realpython.com/python-with-statement/

CSV file I/O

read data in a CSV file

opens file
for reading

```
with open('sinit.csv', 'r') as fp:
    csvreader = csv.reader(fp, delimiter=',')
    for row in csvreader:    iterates rows of file
        Sinit = row
```

`row` is a list of strings (that were separated by the delimiter)

CSV file I/O

```
Nsubj = len(Sinit)
Ncond = 2
Acc = np.zeros((Ncond, Nsubj))
Rts = np.zeros((Ncond, Nsubj))
```

Best Practices

meaningful variable names

parameterize dimensions of arrays rather than hard-code them

```
with open('acc.csv', 'r') as fp:
    csvreader = csv.reader(fp, delimiter=',')
    for i, row in enumerate(csvreader):
        for j in range(Nsubj):
            Acc[i,j] = float(row[j])
```

CSV file I/O

```
Nsubj = len(Sinit)
Ncond = 2
Acc = np.zeros((Ncond, Nsubj))
Rts = np.zeros((Ncond, Nsubj))

with open('acc.csv', 'r') as fp:
    csvreader = csv.reader(fp, delimiter=',')
    for i, row in enumerate(csvreader):
        for j in range(Nsubj):
            Acc[i,j] = float(row[j])
```

CSV file I/O

```
Nsubj = len(Sinit)
Ncond = 2
Acc = np.zeros((Ncond, Nsubj))
Rts = np.zeros((Ncond, Nsubj))

with open('acc.csv', 'r') as fp:
    csvreader = csv.reader(fp, delimiter=',')
    i = 0
    for row in csvreader:
        for j in range(Nsubj):
            Acc[i,j] = float(row[j])
        i += 1
```

this all does
the same thing

CSV file I/O

```
Nsubj = len(Sinit)
Ncond = 2
Acc = np.zeros((Ncond, Nsubj))
Rts = np.zeros((Ncond, Nsubj))

with open('acc.csv', 'r') as fp:
    csvreader = csv.reader(fp, delimiter=',')
    for i, row in enumerate(csvreader):
        for j in range(Nsubj):
            Acc[i,j] = float(row[j])
```

`enumerate()` returns the index and the next item

```
with open('acc.csv', 'r') as fp:
```

opens/closes file gracefully

```
    csvreader = csv.reader(fp, delimiter=',')
```

creates an "iterator" (like a little engine) that will read file line-by-line

```
    for i, row in enumerate(csvreader):
```

```
0.89,0.91,0.78,0.82,0.74,0.81,0.88,0.92,0.83
0.71,0.73,0.81,0.73,0.76,0.83,0.7,0.72,0.83
```

first pass through, `row` contains the first line of the file

```
        for j in range(Nsubj):
            Acc[i,j] = float(row[j])
```

```
0.89,0.91,0.78,0.82,0.74,0.81,0.88,0.92,0.83
0.71,0.73,0.81,0.73,0.76,0.83,0.7,0.72,0.83
```

`float(row[j])` returns `j`th element on the row as a float

`Acc[i,j]` is building a numpy array containing accuracy data indexed by subject `i` and condition `j`

```
with open('acc.csv', 'r') as fp:
```

opens/closes file gracefully

```
    csvreader = csv.reader(fp, delimiter=',')
```

creates an "iterator" (like a little engine) that will read file line-by-line

```
    for i, row in enumerate(csvreader):
```

```
0.89,0.91,0.78,0.82,0.74,0.81,0.88,0.92,0.83
0.71,0.73,0.81,0.73,0.76,0.83,0.7,0.72,0.83
```

first pass through, `row` contains the first line of the file

```
        for j in range(Nsubj):
            Acc[i,j] = float(row[j])
```

```
0.89,0.91,0.78,0.82,0.74,0.81,0.88,0.92,0.83
0.71,0.73,0.81,0.73,0.76,0.83,0.7,0.72,0.83
```

`float(row[j])` returns `j`th element on the row as a float

`Acc[i,j]` is building a numpy array containing accuracy data indexed by subject `i` and condition `j`

```
with open('acc.csv', 'r') as fp:
```
opens/closes file gracefully

```
    csvreader = csv.reader(fp, delimiter=',')
```
creates an "iterator" (like a little engine) that will read file line-by-line

```
    for i, row in enumerate(csvreader):
```

```
0.89,0.91,0.78,0.82,0.74,0.81,0.88,0.92,0.83
0.71,0.73,0.81,0.73,0.76,0.83,0.7,0.72,0.83
```
first pass through, `row` contains the first line of the file

```
        for j in range(Nsubj):
            Acc[i,j] = float(row[j])
```

```
0.89,0.91,0.78,0.82,0.74,0.81,0.88,0.92,0.83
0.71,0.73,0.81,0.73,0.76,0.83,0.7,0.72,0.83
```
`float(row[j])` returns `j`th element on the row as a float

`Acc[i,j]` is building a numpy array containing accuracy data indexed by subject `i` and condition `j`

CSV file I/O

can write out all the data structures to one file

```
with open('data.csv', 'w') as fp:
    csvwriter = csv.writer(fp)
    csvwriter.writerow(sinit)
    csvwriter.writerows(acc)
    csvwriter.writerows(rts)
```

CSV file I/O

data.csv — Edited

```
TP,JA,WO,WM,KL,WS,BB,EC,VB
0.89,0.91,0.78,0.82,0.74,0.81,0.88,0.92,0.83
0.71,0.73,0.81,0.73,0.76,0.83,0.7,0.72,0.83
451,515,574,614,550,643,634,514,613
723,689,712,613,812,743,690,719,772
```

file needs to be read row by row
and parsed into the right data structures
with the right types

CSV file I/O

need to read them in the same way as written out

```
with open('data.csv', 'r') as fp:
    csvreader = csv.reader(fp, delimiter=',')
    Sinit = next(csvreader)

    Nsubj = len(Sinit)
    Ncond = 2
    Acc = np.zeros((Ncond, Nsubj))
    Rts = np.zeros((Ncond, Nsubj))

    for i in range(Ncond):
        row = next(csvreader)
        for j in range(Nsubj):
            Acc[i,j] = float(row[j])

    for i in range(Ncond):
        row = next(csvreader)
        for j in range(Nsubj):
            Rts[i,j] = float(row[j])
```

if file other than CSV format, you would not use `csv.reader()`

`next()` returns the next item of an iterator

Homework 4

Q1 first **(a)** asks you to save HDR(t) and t (plus some other things) from Homework 3* as a CVS file (in a particular format) using the techniques I just went over and second **(b)** asks you to read in that CSV file into the appropriate variables and plot it (using your code from Homework 3*).

Your CSV file should have the follow structure:
1st line should be an informative note (< 50 characters) about what the file contains.
2nd line is the <u>number of time steps</u> in *HDR(t)*.
3rd line should be the <u>names</u> of the parameters of the HDR (from Homeworks 2 and 3), separated by commas.
4th line should be the <u>values</u> of the parameters (from Homeworks 2 and 3), separated by commas.
The remaining lines should be <u>each value of *t* and its corresponding *HDR(t)*</u>, separated by commas (in other words, if you had 1000 values of *t* and *HDR(t)* in your numpy arrays, these should be 1000 lines in the CSV file).

* if your code from Homework 3 did not work, you should correct it based on the comments from Jason (seeing me or Jason if you need help)

an example of reading and analyzing some behavioral data using numpy arrays

operations on numpy arrays

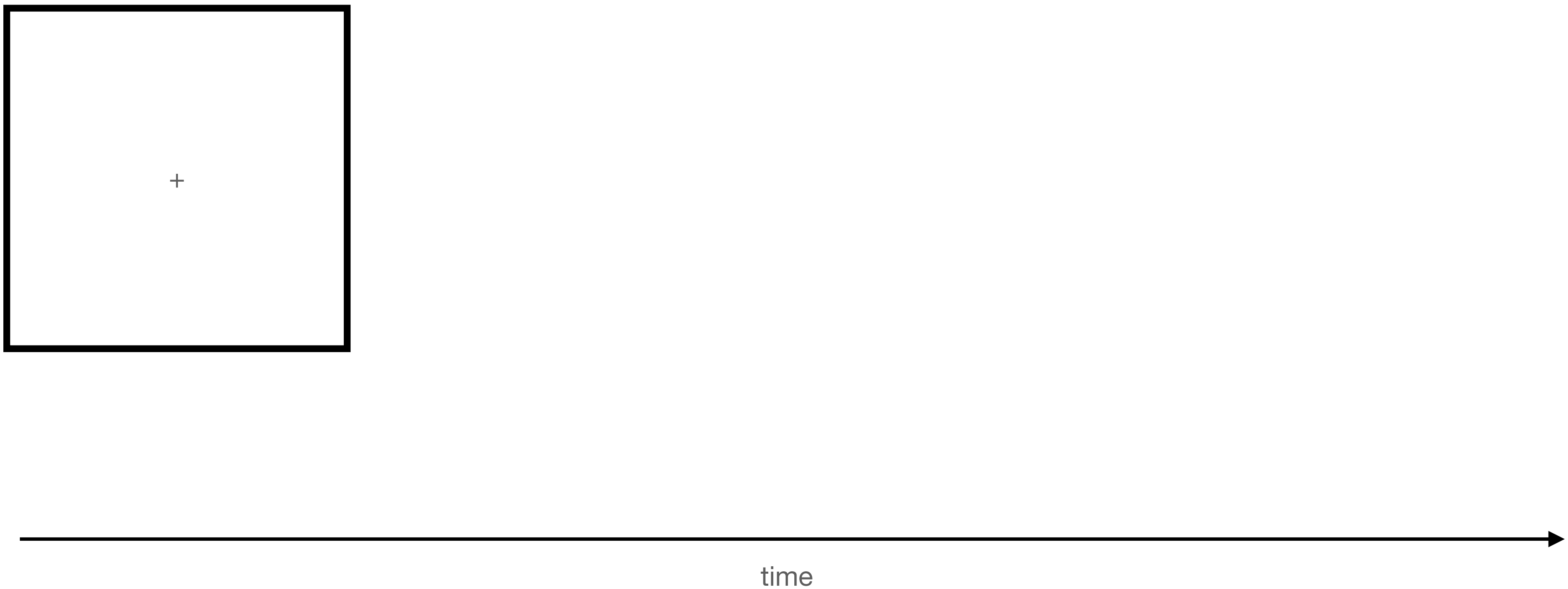
numpy arrays contain "data" (of the same type); in the context of psychology and neuroscience, these could be

- behavioral data (e.g., choices, response times, as a function of subjects, conditions, trials) from an experiment
- 1D signal (time-series) (e.g., intracellular, extracellular, scalp voltage as a function of time) from neural recordings
- 2D signal (time-series) (e.g., N electrical channels x T time steps), 3D signal (e.g., S subjects x N channels x T time)
- 3D eye movements with (x, y) gaze direction by time
- 2D B/W image with (x, y) intensity, 3D color images with (x,y) intensity for R, G, and B color channels
- 3D functional or anatomical brain scan (x, y, z)
- 4D functional or anatomical brain scan (x, y, z) by time steps

is there a T?

a simple experiment (visual search)

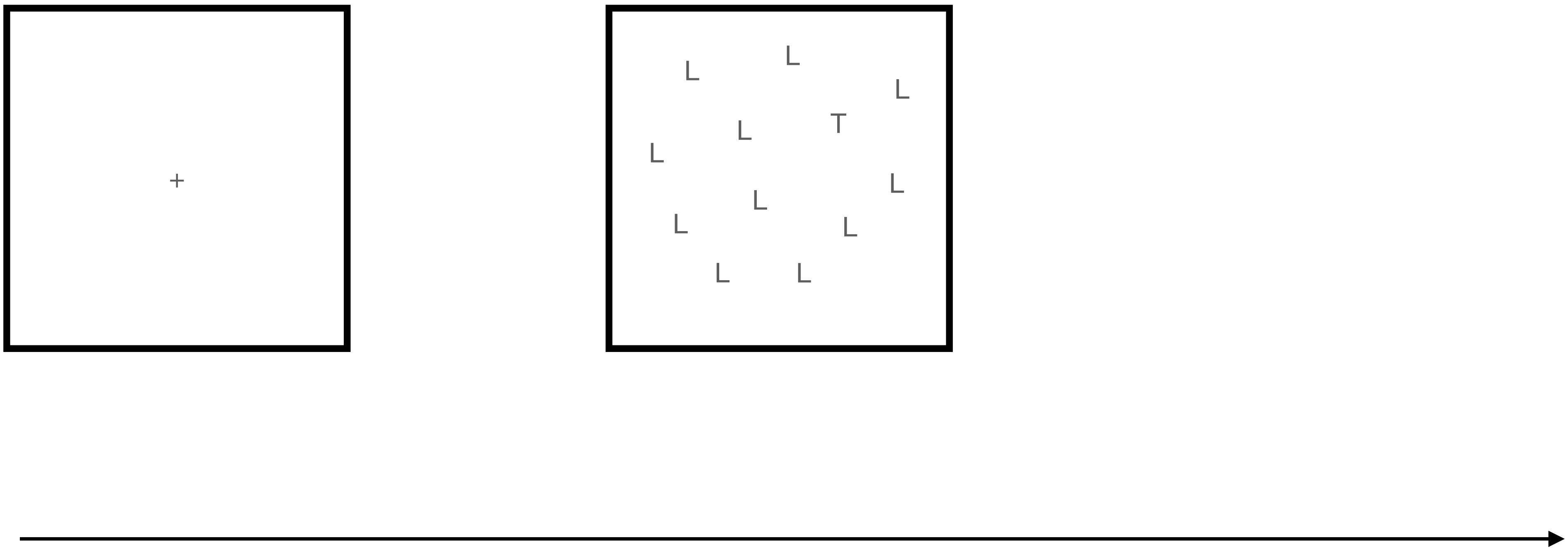
Trial 1



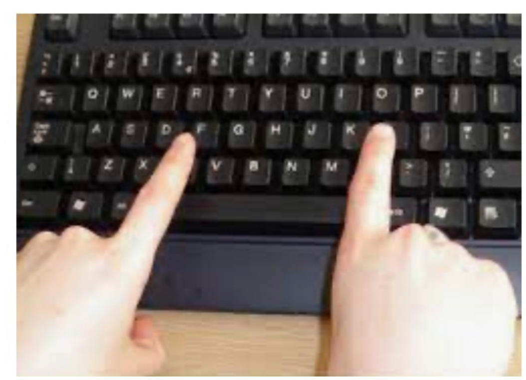
is there a T?

a simple experiment (visual search)

Trial 1



time



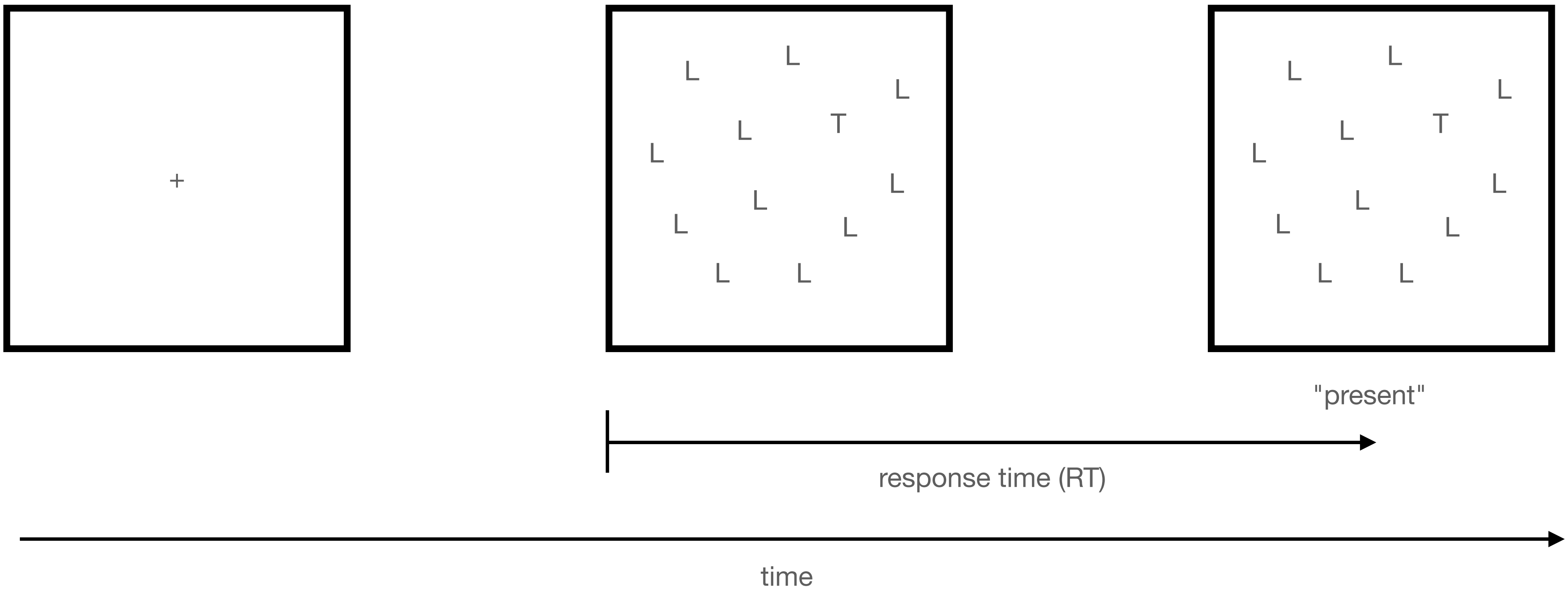
is there a T?

a simple experiment (visual search)

Trial 1

target present condition

correct



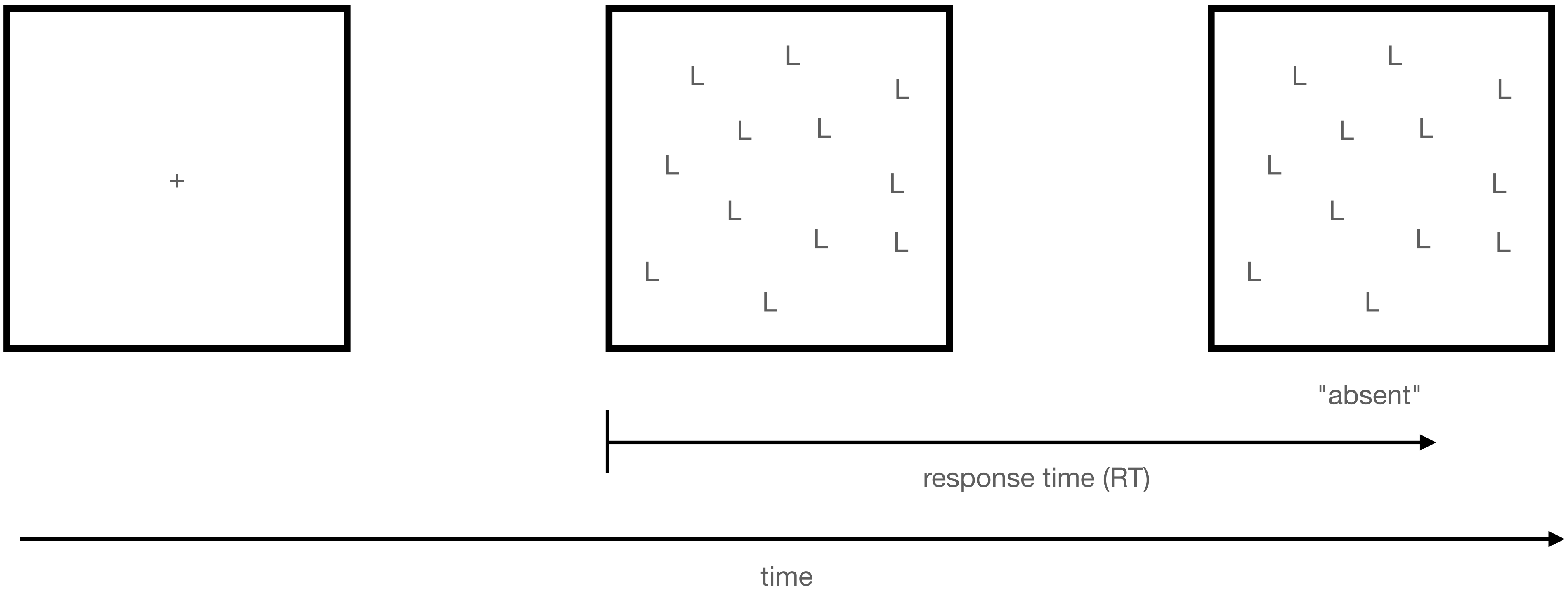
is there a T?

a simple experiment (visual search)

Trial 2

target absent condition

correct



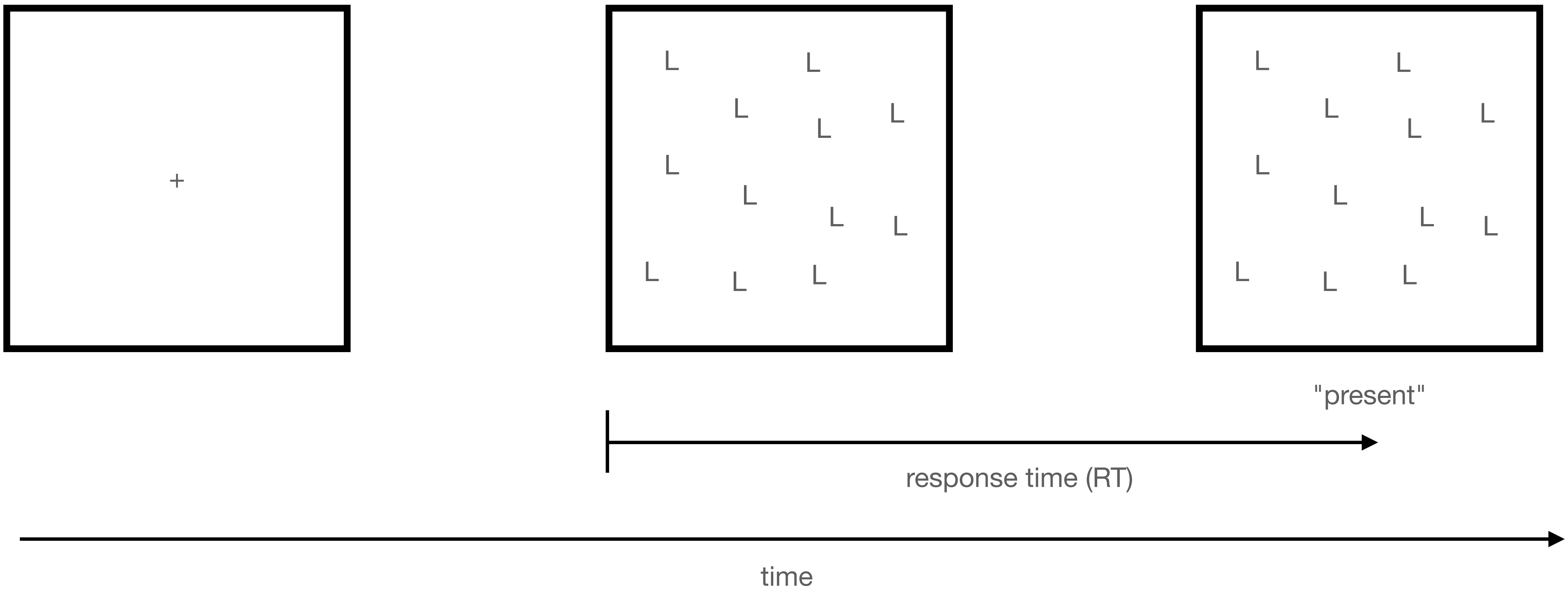
is there a T?

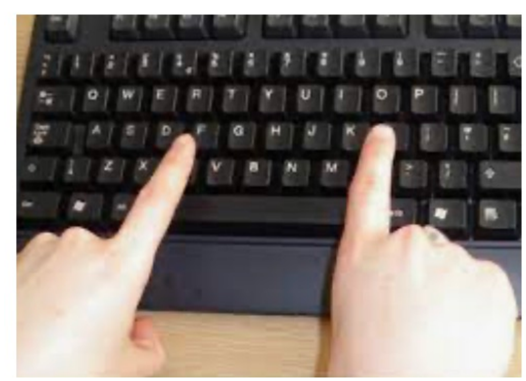
a simple experiment (visual search)

Trial 3

target absent condition

error





(simulated) data from an experiment with 2 conditions

`difdata.csv` (ASCII text file in CSV format) **one subject's data**

each line (except first)

correct (1)
error (0)

trial #

$\boxed{5},\boxed{1},\boxed{0},\boxed{0.416}$

condition (1 or 2)
present absent

response time (in sec)

trials →

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

how to write code that processes this file

`difdata.csv` (ASCII text file)

(0) get csv

(1) open file for reading

each line (except first)

trial #, condition (1 or 2), correct (1) error (0), response time (in sec)

$\boxed{5},\boxed{1},\boxed{0},\boxed{0.416}$

trials → 1000

(2) read # trials

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

how to write code that processes this file

`difdata.csv` (ASCII text file)

(0) get csv

(1) open file for reading

each line (except first)

trial #   correct (1) error (0)

```
5,1,0,0.416
```

condition (1 or 2)   response time (in sec)

trials

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

(2) read # trials

(3) read each line, filling arrays with condition, response, RT

later, we'll do more

see `FileIO.ipynb`

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

icondition ichoice iRT

```python
# read in the data

import csv
import numpy as np

with open('difdata.csv', 'r') as fp:
    # create the "reader" object
    csvreader = csv.reader(fp, delimiter=',')

    # get a line of the file
    row = next(csvreader)

    # that first line is the # trials
    Ntrials = int(row[0])

    # using # trials, preallocate np arrays to hold condition, choice, and RT
    icondition = np.zeros(Ntrials, dtype=int)
    ichoice    = np.zeros(Ntrials, dtype=int)
    iRT        = np.zeros(Ntrials, dtype=float)

    # loop over all trials (all remaining lines in the file)
    for i, row in enumerate(csvreader):
        icondition[i] = int(row[1])
        ichoice[i]    = int(row[2])
        iRT[i]        = float(row[3])
```

Homework 4

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

iRT

ichoice

icondition

(a) (6 points) Following the discussion from class, I want you to partition the data so that one two-dimensional array that holds the choices in condition 1 and the choices in condition 2 and another two-dimensional array that holds the RTs in condition 1 and the RTs in condition 2. **First**, do this using for loops. **Second**, do this using logical (Boolean) indexing.

one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

logical (Boolean) indexing

```
a = np.array([1, 2, 3, 4, 5])
b = np.array([False, True, False, False, True])
print(a[b])
```

a and b need to be the same size

```
a = np.arange(100)
b = (a % 2) == 0
print(a[b])
```

what does this do?

```
print(a[((a % 2) == 0)])
```

this is the same

https://numpy.org/doc/stable/user/basics.indexing.html#boolean-or-mask-index-arrays see `LogicalIndexing.ipynb`

Homework 4

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

iRT

ichoice

icondition

one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

Homework 4

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

icondition ichoice iRT

for **Homework 4**, write code that creates a 2x500 array for choice and RT data, separated by experimental condition

choice

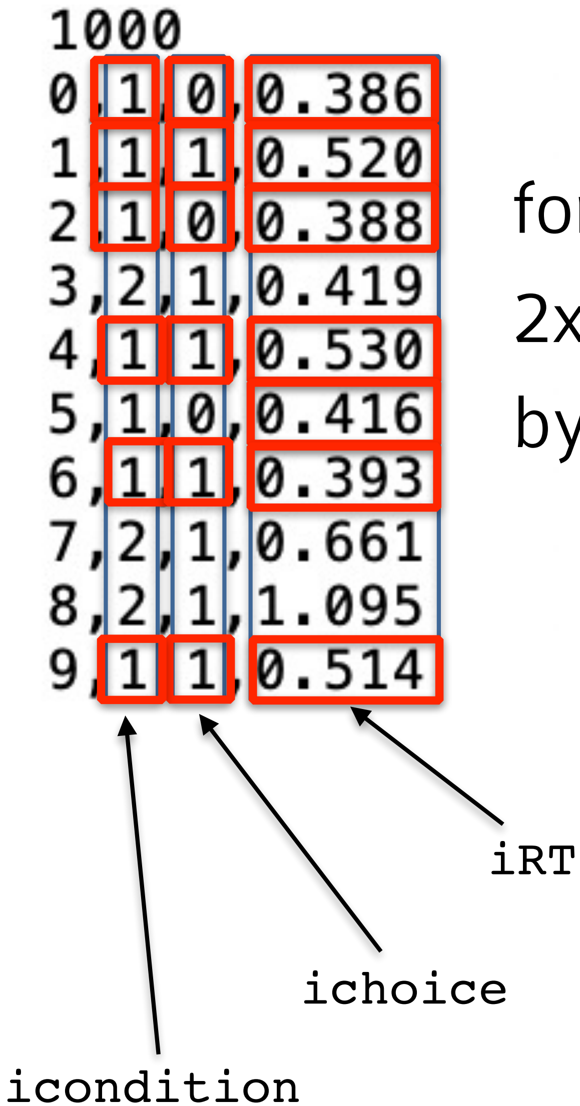
0	1	0	1	0	1	1	1	. . .
1	1	1	0	0	1	1	0	

RT

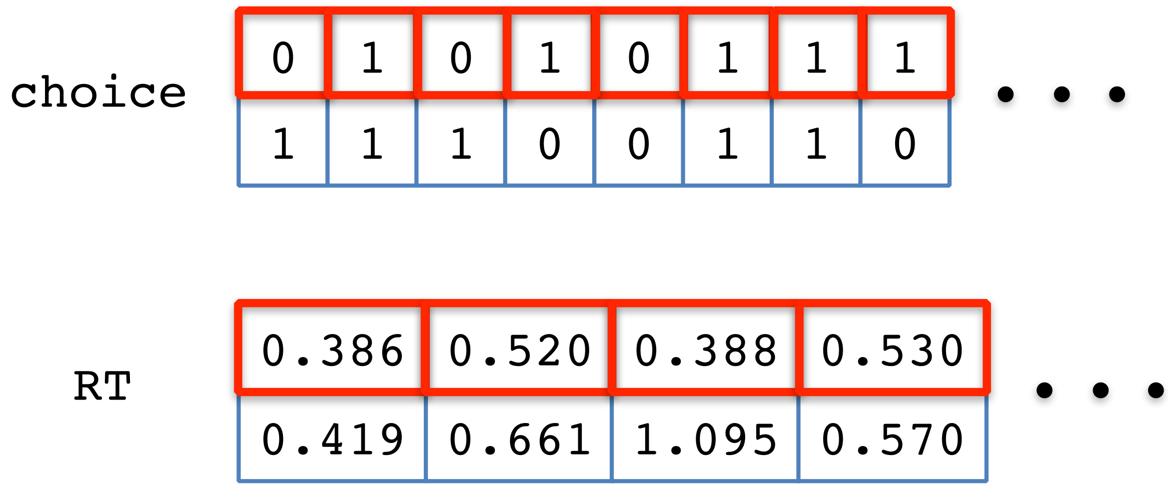
0.386	0.520	0.388	0.530	. . .
0.419	0.661	1.095	0.570	

one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

Homework 4



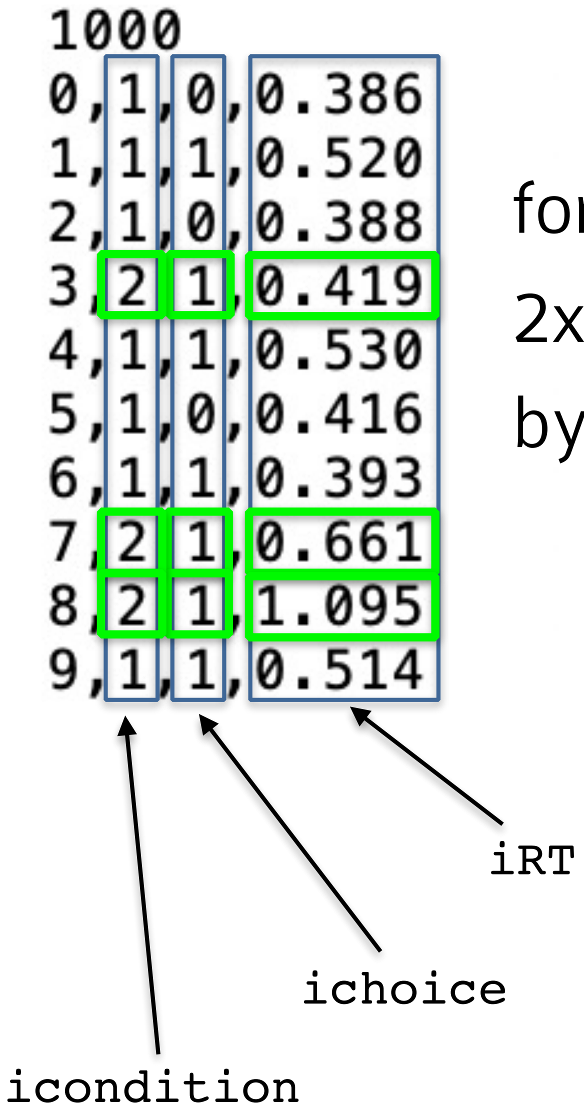
for **Homework 4**, write code that creates a 2x500 array for choice and RT data, separated by experimental condition

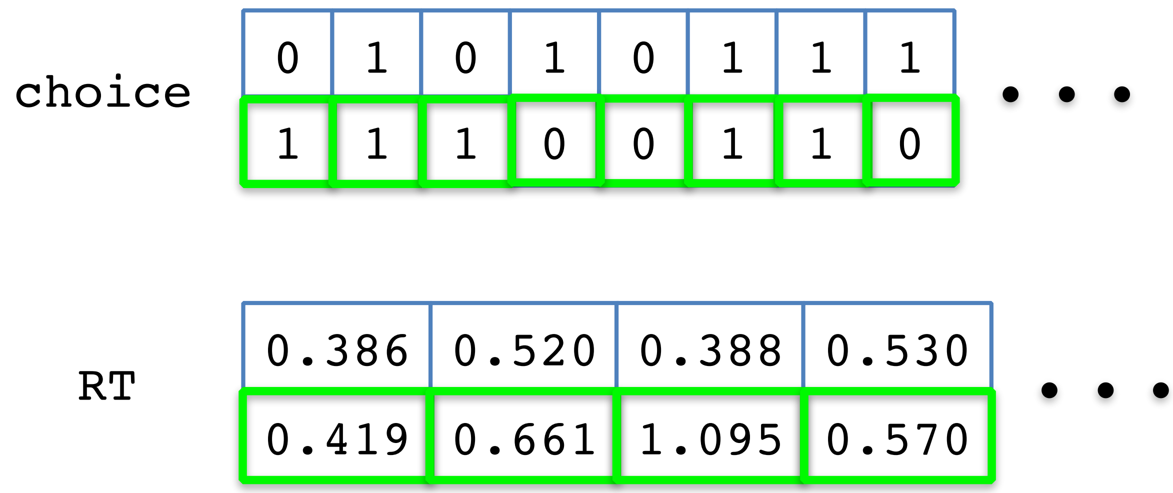


one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

Homework 4

for **Homework 4**, write code that creates a 2x500 array for choice and RT data, separated by experimental condition



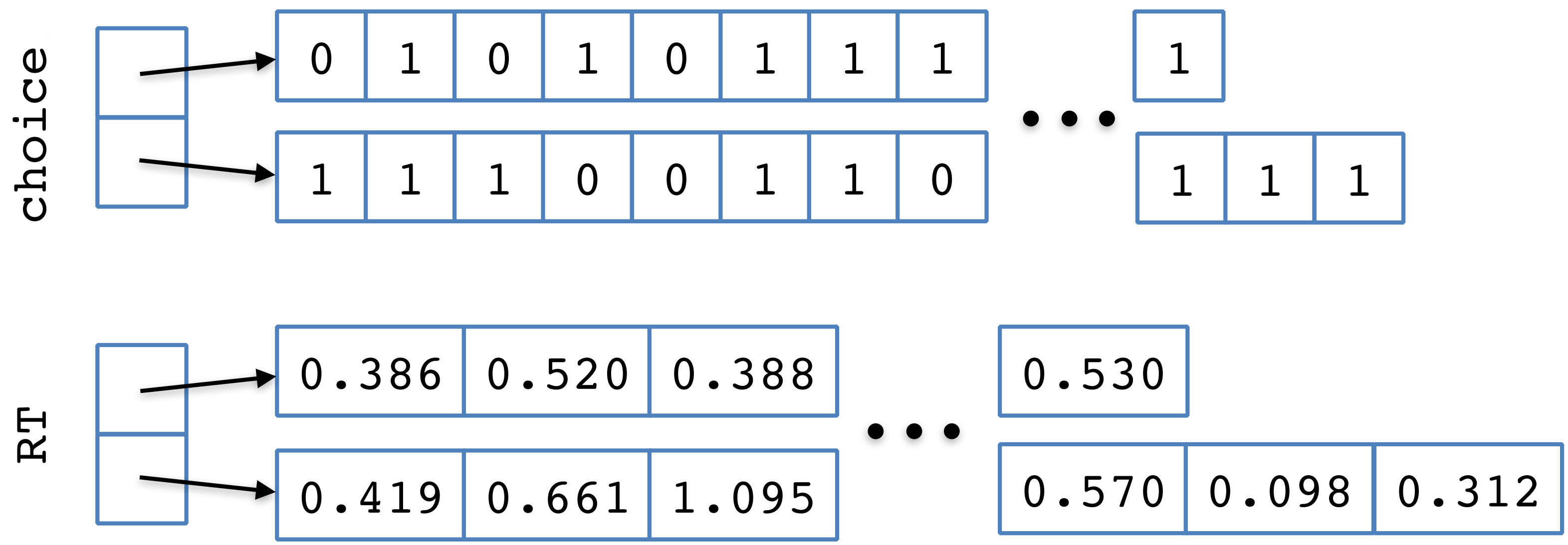


one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

Homework 4

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

how would you remove "outliers", for example trials where RT was outside some bound (e.g., RT<0.100 or RT>1.000)



one thing we might want to do is divide up the choice and RT data by condition, creating separate arrays for condition 1 and for condition 2 - how to do that?

```
1000
0,1,0,0.386
1,1,1,0.520
2,1,0,0.388
3,2,1,0.419
4,1,1,0.530
5,1,0,0.416
6,1,1,0.393
7,2,1,0.661
8,2,1,1.095
9,1,1,0.514
```

Homework 4

(b) (6 points) Following the discussion from class, I want you to remove "outliers" based on RT, in this case trials where RT is outside some bound (RT<0.100 or RT>1.000). This will result in a list of arrays for the choices with outlier trials removed and a list of arrays for the RTs with the outlier trials removed. **First**, do this using for loops. **Second**, do this using logical (Boolean) indexing. Remember from discussion in class that here you will not be able to use a 2x500 numpy array because the number of resulting trials after removing outliers will be unequal (instead, use a list of numpy arrays).

Command-Line Input

command-line input

input a string

```
name = input('Enter Name :’)
```

entering a non-string requires a type conversion

```
SubjN = int(input('Enter the Subject Number: '))
SessN = int(input('Enter the Session Number: '))
```