CSV Editing With Python (and Pandas)

For Non-Programmers!

Presentation Goals

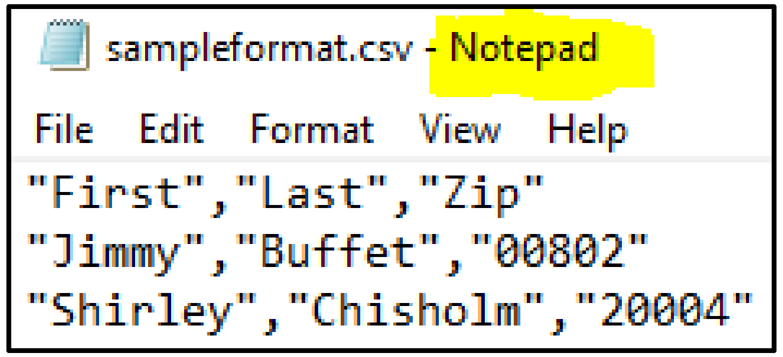
- Make Python code look accessible to people who often say:

“I have **no idea why that works**, but
I’ll copy+edit it anyway
if it does the job.”

- Demonstrate cool code you’ll want to ~~*break*~~ try

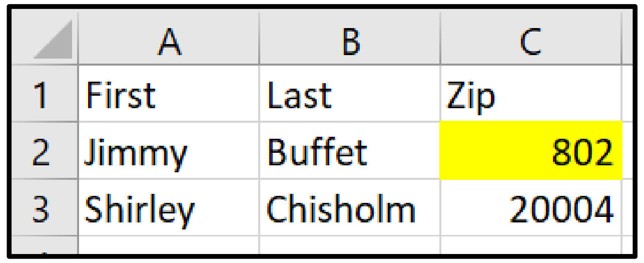
Basics

- CSV = Comma Separated Values
  - Text-editor-friendly
  - No formatting
  - Database export/import



```
"First","Last","Zip"
"Jimmy","Buffet","00802"
"Shirley","Chisholm","20004"
```

- "Table-shaped" data, so Excel often easy
  - But sometimes not … so … **Hi!** ☺



	A	B	C
1	First	Last	Zip
2	Jimmy	Buffet	802
3	Shirley	Chisholm	20004

Python + Pandas

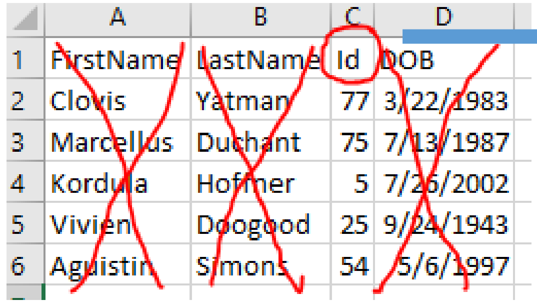
- **Python:** programming language

- **Pandas:** module *(plugin)* for Python
  - Adds CSV-related commands

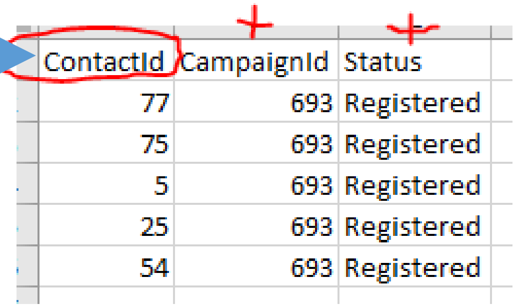
- Programs run in an **IDE**
  - **IDE:** code-editing software with a run button

Use Excel

- Simple column manipulation & fills:

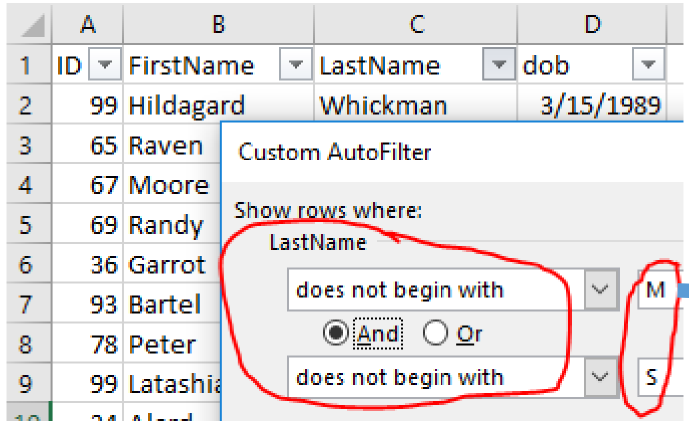


	A	B	C	D
1	FirstName	LastName	Id	DOB
2	Clovis	Yatman	77	3/22/1983
3	Marcellus	Duchant	75	7/13/1987
4	Kordula	Hoffner	5	7/26/2002
5	Vivien	Doogood	25	9/24/1943
6	Aguistin	Simons	54	5/6/1997



ContactId	CampaignId	Status
77	693	Registered
75	693	Registered
5	693	Registered
25	693	Registered
54	693	Registered

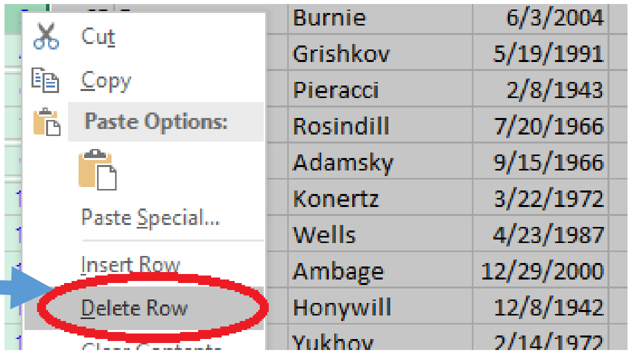
- Simple "filter-and-delete-rows":



	A	B	C	D
1	ID	FirstName	LastName	dob
2	99	Hildagard	Whickman	3/15/1989
3	65	Raven		
4	67	Moore		
5	69	Randy		
6	36	Garrot		
7	93	Bartel		
8	78	Peter		
9	99	Latashia		

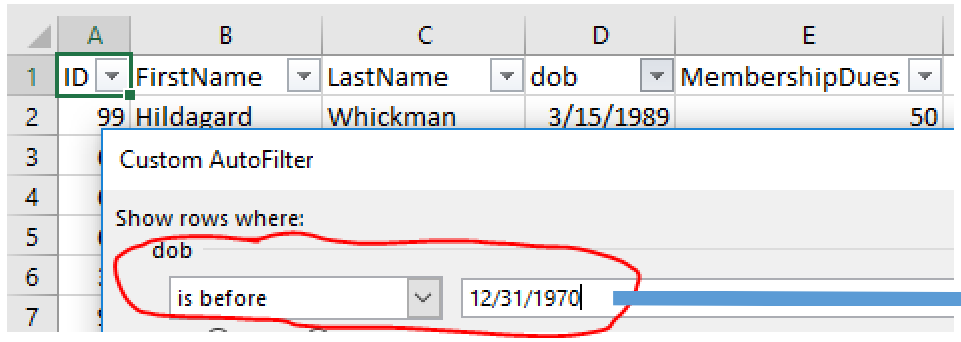
Custom AutoFilter

Show rows where:
LastName
does not begin with M
[✓ And] Or
does not begin with S



Burnie	6/3/2004
Grishkov	5/19/1991
Pieracci	2/8/1943
Rosindill	7/20/1966
Adamsky	9/15/1966
Konertz	3/22/1972
Wells	4/23/1987
Ambage	12/29/2000
Honywill	12/8/1942
Yukhov	2/14/1972

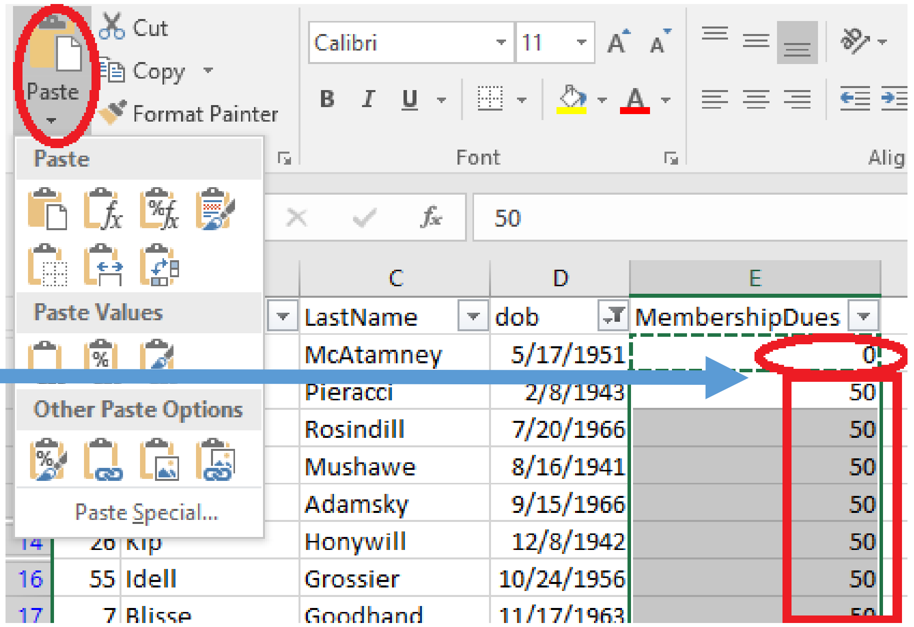
- Simple "filter and edit value":



	A	B	C	D	E
1	ID	FirstName	LastName	dob	MembershipDues
2	99	Hildagard	Whickman	3/15/1989	50

Custom AutoFilter

Show rows where:
dob
is before 12/31/1970



50

		C	D	E
		LastName	dob	MembershipDues
		McAtamney	5/17/1951	0
		Pieracci	2/8/1943	50
		Rosindill	7/20/1966	50
		Mushawe	8/16/1941	50
		Adamsky	9/15/1966	50
14	26 Kip	Honywill	12/8/1942	50
16	55 Idell	Grossier	10/24/1956	50
17	7 Blisse	Goodhand	11/17/1963	50

Use Python

- Filter and-delete-rows …
  …with 1-million-row table that freezes Excel

- Filter and edit value …
  … 50 times in a row with different variations

- Pivot & filter the pivoted data, e.g.
  - Delete all rows except the oldest member of a household

- VLOOKUP against multiple columns, e.g.
  - Combine everyone from 2 spreadsheets with the same first name, last name, and phone number

OK to combine!

- Excel: exploration
- Python: automation

- Example: 100,000 rows, no idea:
  - # of rows with an "inter-column data mismatch"
  - Categories of "mismatch" they would cluster into
    - ("Do I care?" "How did it get this way?")

6 hours of exploring & thinking.

Had to **start over** with a fresh copy of the data **halfway through**.

Had I not "scripted" my work, would have been **9 hours**.

1. Python:
   - Add a blank "**MismatchType**" **column**
   - **Delete rows** with no mismatch
2. Excel:
   - **Play** with filters to discover mismatch "categories" in remaining rows
3. Python:
   - For each "mismatch category" discovered:
     - Label such rows under "**MismatchType**"
     - **Delete rows** I consider unimportant mismatches
4. Repeat steps 2-3 until every row has a "**MismatchType**" value or is gone
5. Excel:
   - **Show** colleagues remaining 1,000 rows clustered into 20 "mismatch types" and discuss

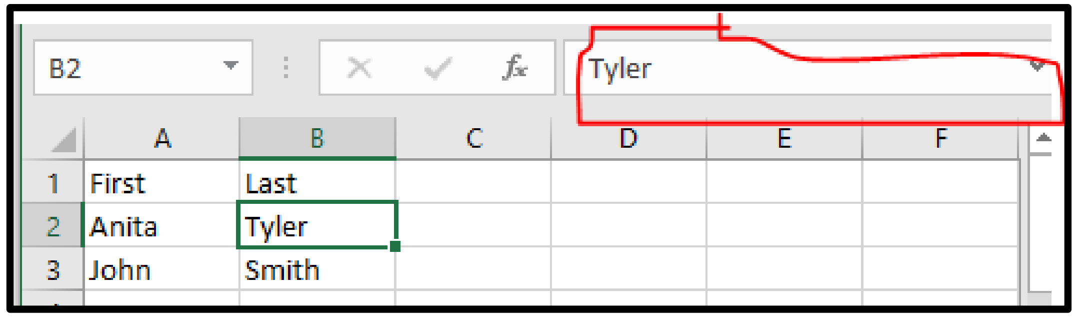
Programming 101

(To help you follow the examples)

What makes a program a program?
Why isn't Excel a program?

Expressions & Statements & Operations

- Expression: **<u>code</u> that *becomes/is* <u>a value</u>**.
  - Nestable
    - 1 + 1
    - "Hello".startsWith("P")
    - 3 * 2.5 * 4 < 1
    - concatenate("h","e","l","l","o")



	A	B	C	D	E	F
1	First	Last				
2	Anita	Tyler				
3	John	Smith				

- Statement: **<u>standalone code</u> that *does* <u>something noticeable</u>**.
  - NOT nestable
    - "**Show me** the value of '1+1' **on my screen**."
    - "**Store** the value of '1+1' **in** a variable called 'myMath'"
    - "**Import** a 'package' **that lets me type a wider range** of commands in my code."

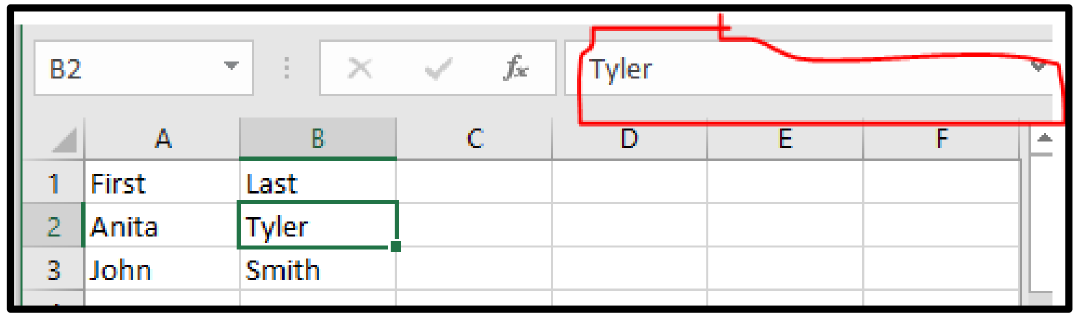
- Operation: **<u>code</u>** that **<u>combines expressions together</u>** into bigger expressions or into a statement
    - +
    - .startsWith(…)
    - <
    - concatenate (… , … , … , …)
    - "show me … on my screen"
    - "store … into a variable called …"

Expression-Nesting Pop Quiz

- “Hello”.startsWith(“P”)
- 3 * 2.5 * 4 < 1

How many expressions can you see in each example above?

Getting really good at this game will help you “backspace & replace” useful code you find on the internet, even if you don’t understand it!


B2 | Tyler

	A	B	C	D	E	F
1	First	Last				
2	Anita	Tyler				
3	John	Smith				

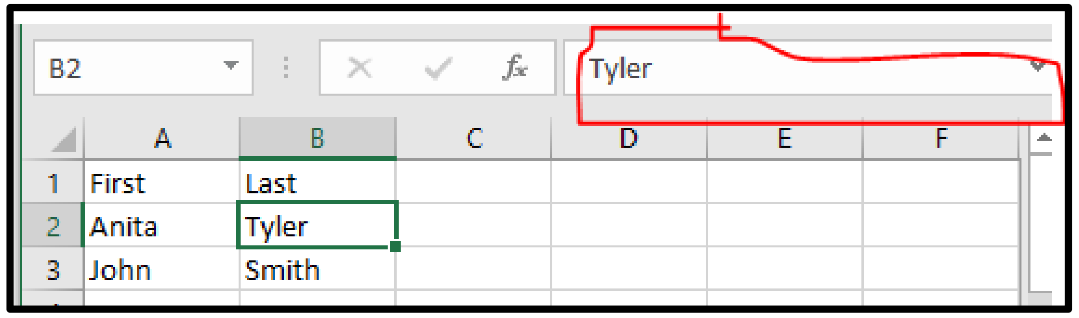
Statements Make Programs

- Statement: **smallest unit of runnable code** in a program

- Multiple statements = a program
  - *(1-statement program possible, like 1-sentence essay)*

- Typically **1 statement per line** of code *(especially in Python)*

Expressions ≠ Programs
∴ Excel ≠ Programs



B2 | Tyler

	A	B	C	D	E	F
1	First	Last				
2	Anita	Tyler				
3	John	Smith				

- Excel: "expressions" only

- Besides Macros/VBA (often a pain), no way to save a sequence of *doing* things.

- ∴ we code!
  - (w/ Python, because ☺)

Programming 101

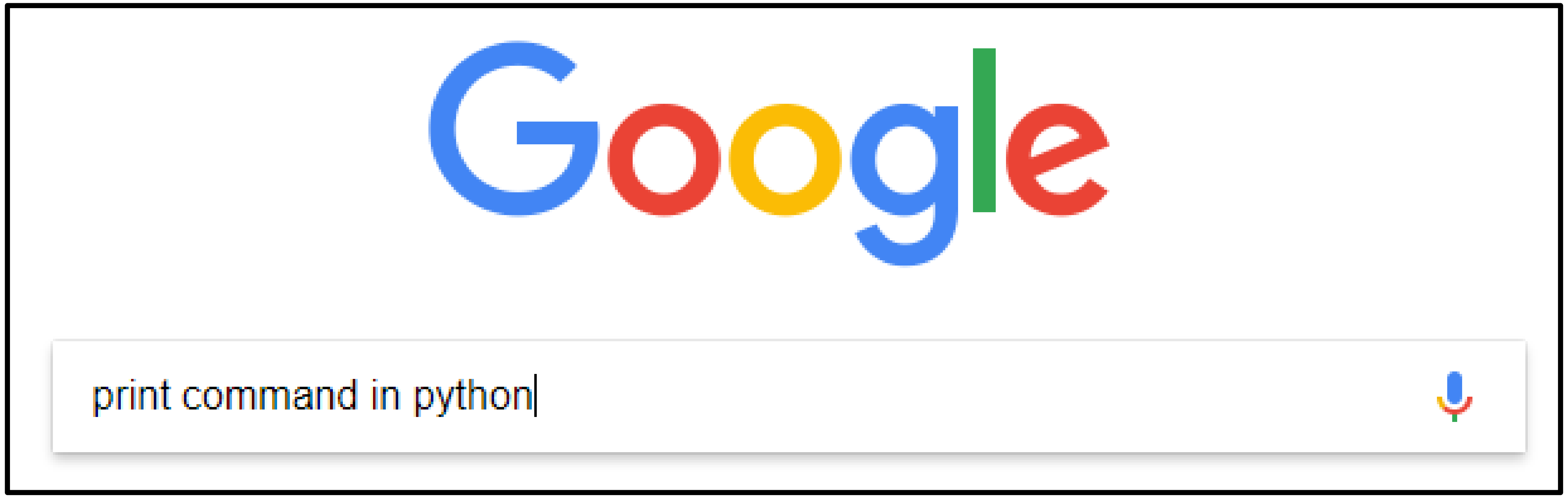
Culture Shock Alleviation

Coding Culture Shock: Not Visual

- Working "blind" (vs. Excel) 😱 😓 😭
- Useful tricks:
  - **"Print" statements**
    (puts otherwise-invisible data on the screen) 😂
  - Nicknaming intermediate "expression" outputs ("**setting variables**") for later use in code
    (like "wet" & "dry" baking bowls)
  - **"Comments"**
    (words in your code that aren't really code – notes to self)

Intro

- No shame in “Programming By Google”



print command in python

Programming 101

Seeing your data like a programmer

Data Types

- Data Type: dimension & kind
  - 0-D (**single points** of data)
    - **Text**? **Number**? True/False (**Boolean**)? Blank (**Null**)?
  - 1-D collections (**lists** of 0-D points)
    - **Row-like** *(meant to represent 1 "record")*?
    - **Column-like** *(meant to represent 1 "field" across multiple records)*?
      - If column-like, what **type** (text/number/Boolean/etc) are the 0-D "data points" **within** this list?
  - 2-D collections (**tables** of 1-D row-lists & 1-D column-lists intersecting at 0-D points)

- Constrains what "<u>**operations**</u>" we can do to data. Can we …

+, - ?	0D #, 0D text if + is "concatenate"
fetch 1st letter?	0D text data
<, == ?	0D number, 0D text …
SELECTION – 1D & 2D data:	fetch "item #3" or "fetch odd-numbered items"?
ITERATION – 1D & 2D data:	do something separately to every item, leaving behind a new value in each item's place? (e.g. multiply each by 3)
AGGREGATION – 1D & 2D data:	combine all the items together into just one value? (e.g. "max" or "sum")

Operations' "Input Expressions"

Operations require different **numbers** & **placement** of "input expressions" ***(You've seen this in Excel!)***

- 0-input example: **NOW()**
  - output = **{current date & time}** (true/false "DateTime"-typed data)

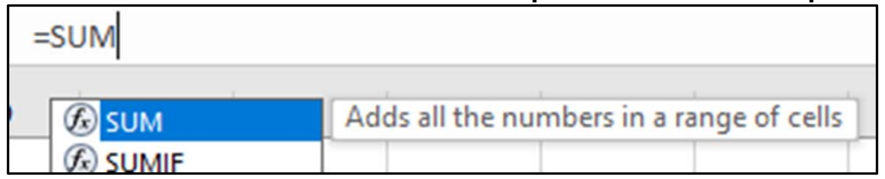
- 1-input example: **ISNUMBER("apple")**
  - input = **"apple"** (text-typed data)
  - output = **False** (true/false "Boolean"-typed data)

- 2-input example: **1 + 4**
  - inputs = **1** & **4** (number-typed data)
  - output = **5** (number-typed data)
- *(Remember: **1 + 5 + 3** is actually two back-to-back two-input operations, **1 + 5** and **6 + 3**.)*

- 3+-input example: **SUM(3,4,5,9,4)**
  - inputs = **3**, **4**, **5**, **9**, & **4** (number-typed data)
  - output = **25** (number-typed data)

♥ Data Types = Easier "Expression" Writing

- Tricky #1: Fewer helpful hints about "expression operations while you program *(in online manuals)*

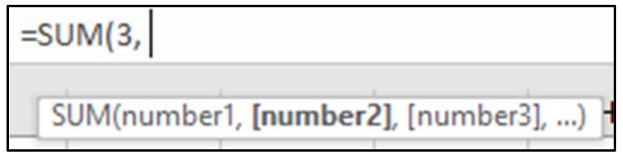


=SUM

SUM

SUMIF

Adds all the numbers in a range of cells



=SUM(3,

SUM(number1, **[number2]**, [number3], ...)

- Tricky #2: Not just "AROUND" & "BETWEEN" operations like **ISNUMBER("apple")** & **1+4**
  - Also "AFTER" operations, connected by a period, like **"Hello".lower()**
  - Worse: "AFTER" operations in Pandas w/ random *extra* period, like **ExpressionHere.str.lower()**

Q: Panic? 😱 😓 😭

A:

- print(ExpressionHere)
- print(type(ExpressionHere))
- CoolVariableName = ExpressionHere
- print(CoolVariableName)
- print(type(CoolVariableName))

Confused what **9 - 4 < 2** does? Inspect smaller problems!
- print(...) & print(type(...)) **3-4**, **1**, **5<1**, **1<2**, or **3<3**.
- Copy/paste back together, like big Excel formulas.

Python Example: “Print” things to read them

print(**'Hello World'**)	Hello World
print(**type('Hello World')**)	<class 'str'>
print(**5**)	5
print(**type(5)**)	<class 'int'>
print(**None**)	None
print(**type(None)**)	<class 'NoneType'>
print(**False**)	False
print(**type(False)**)	<class 'bool'>
print(**3 * 2.5 * 4**)	30.0
print(**type(3 * 2.5 * 4)**)	<class 'float'>
print(**3 * 2.5 * 4 < 1**)	False
print(**type(3 * 2.5 * 4 < 1)**)	<class 'bool'>
myFirstVariable = **3 * 2.5 * 4**	*{{{{{nothing prints out for this line}}}}}*
print(**myFirstVariable**)	30.0
print(**type(myFirstVariable)**)	<class 'float'>
print(**myFirstVariable < 1**)	False
print(**type(myFirstVariable < 1)**)	<class 'bool'>
print(**'Bye!'**)	Bye!

Programming 101

“Grammar” Gotcha: “=“ vs. “==”

== vs. = 😱 😓 😭

- ==
  - **expression operation** meaning:

    True/False: does the left side equal the right side?

    - "**1+2 == 4-1**" is an "expression" whose "output value" is "**True**"

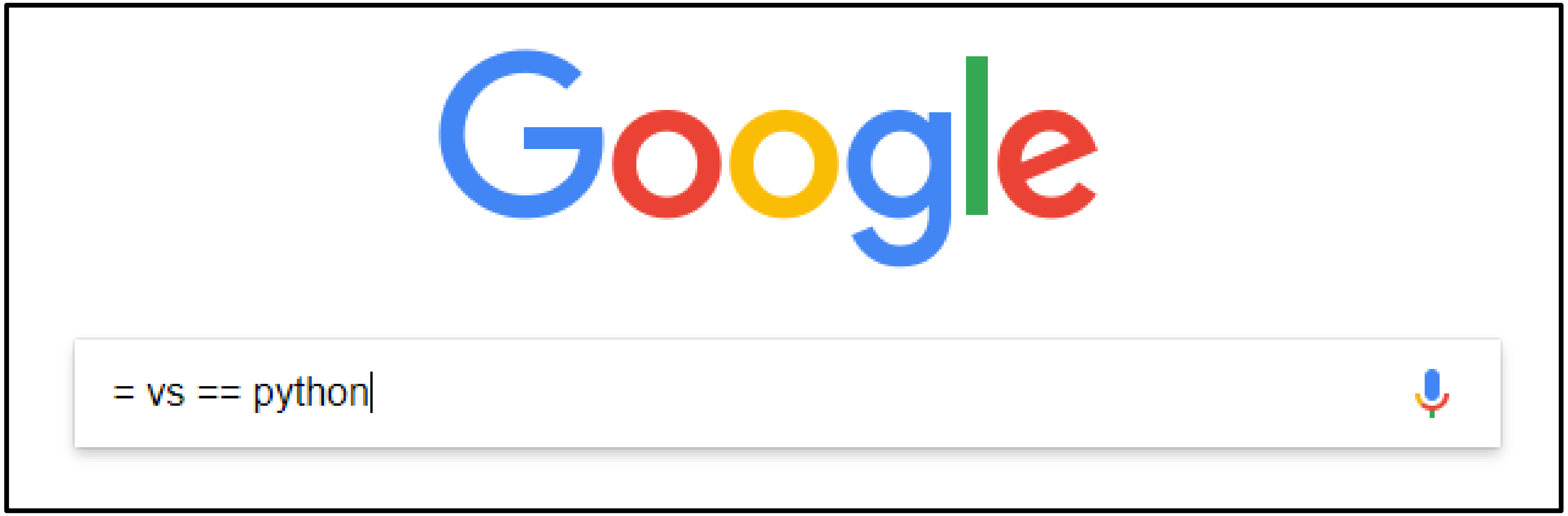
- =
  - **statement operation** meaning

    save the 'output value' of the 'expression' to the right of the '='
    under the nickname mentioned to the left of the '='

    - "**equalityCheckResult = 1+2 == 4-1**"
      is a "statement" that saves "**True**" into "**equalityCheckResult**"

Reminder

- No shame in “Programming By Google”!



= vs == python

Examples

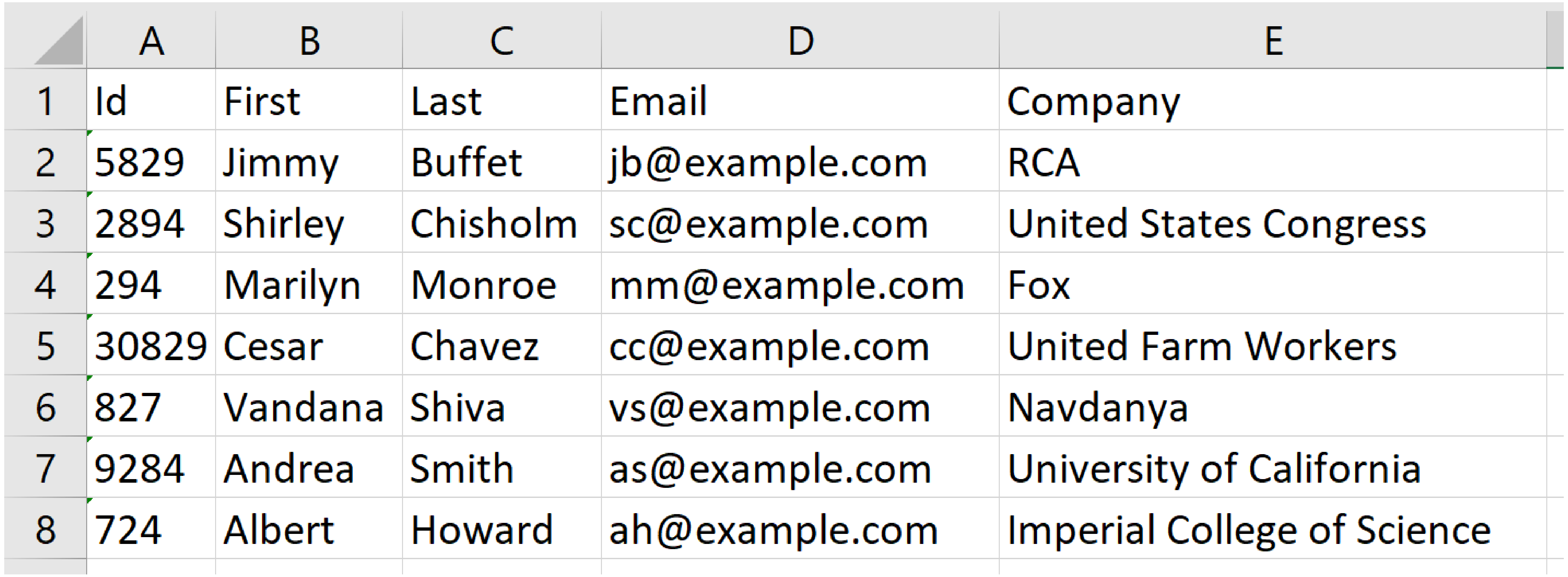
Enjoy the code, but
(40 minutes ≠ expert!)
Watch the input→output data

Runnable code:

https://pypancsv.github.io/pypancsv

sample1.csv

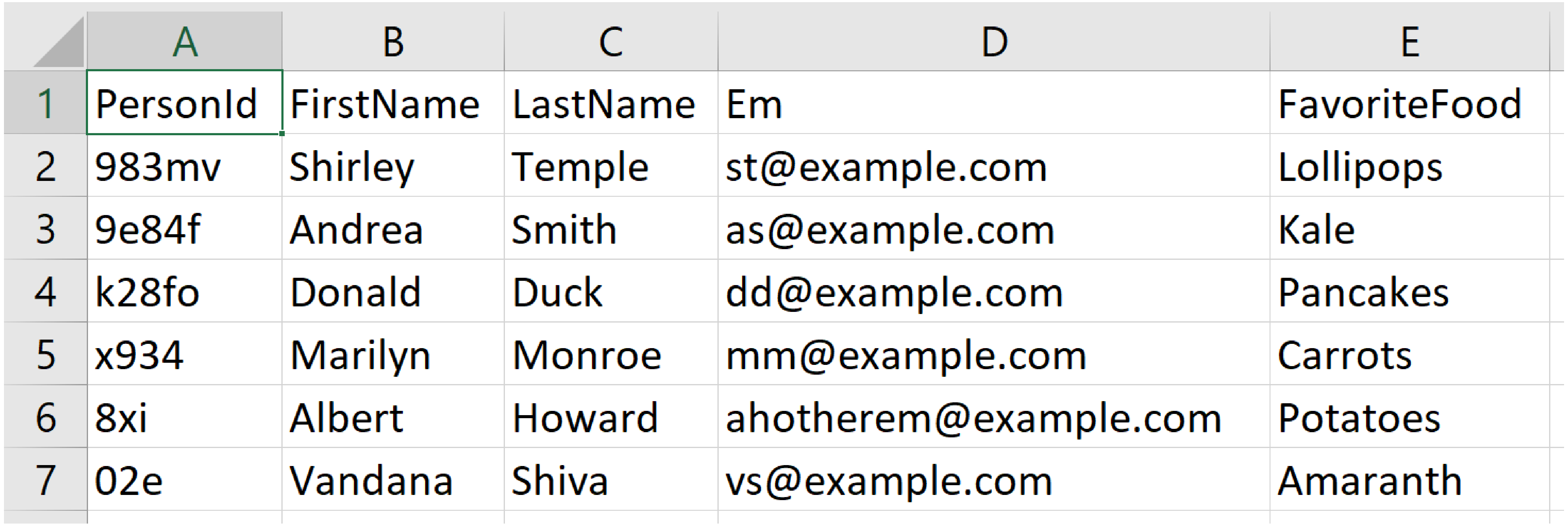
- 7 rows, 5 columns (people & **employer**)
- Contacts from "Data Source **#1**"



	A	B	C	D	E
1	Id	First	Last	Email	Company
2	5829	Jimmy	Buffet	jb@example.com	RCA
3	2894	Shirley	Chisholm	sc@example.com	United States Congress
4	294	Marilyn	Monroe	mm@example.com	Fox
5	30829	Cesar	Chavez	cc@example.com	United Farm Workers
6	827	Vandana	Shiva	vs@example.com	Navdanya
7	9284	Andrea	Smith	as@example.com	University of California
8	724	Albert	Howard	ah@example.com	Imperial College of Science

sample2.csv

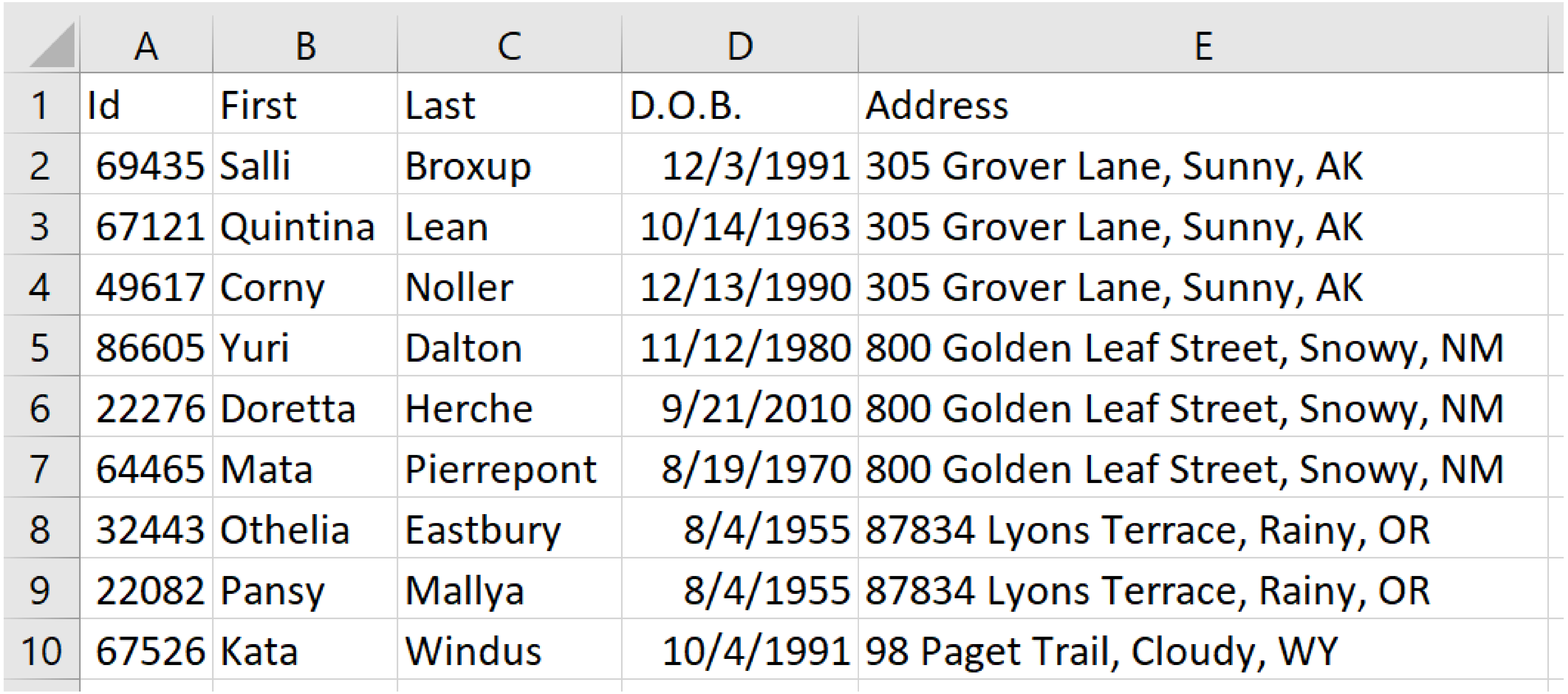
- 6 rows, 5 columns (people & **favorite food**)
- Contacts from "Data Source **#2**"



	A	B	C	D	E
1	PersonId	FirstName	LastName	Em	FavoriteFood
2	983mv	Shirley	Temple	st@example.com	Lollipops
3	9e84f	Andrea	Smith	as@example.com	Kale
4	k28fo	Donald	Duck	dd@example.com	Pancakes
5	x934	Marilyn	Monroe	mm@example.com	Carrots
6	8xi	Albert	Howard	ahotherem@example.com	Potatoes
7	02e	Vandana	Shiva	vs@example.com	Amaranth

sample3.csv

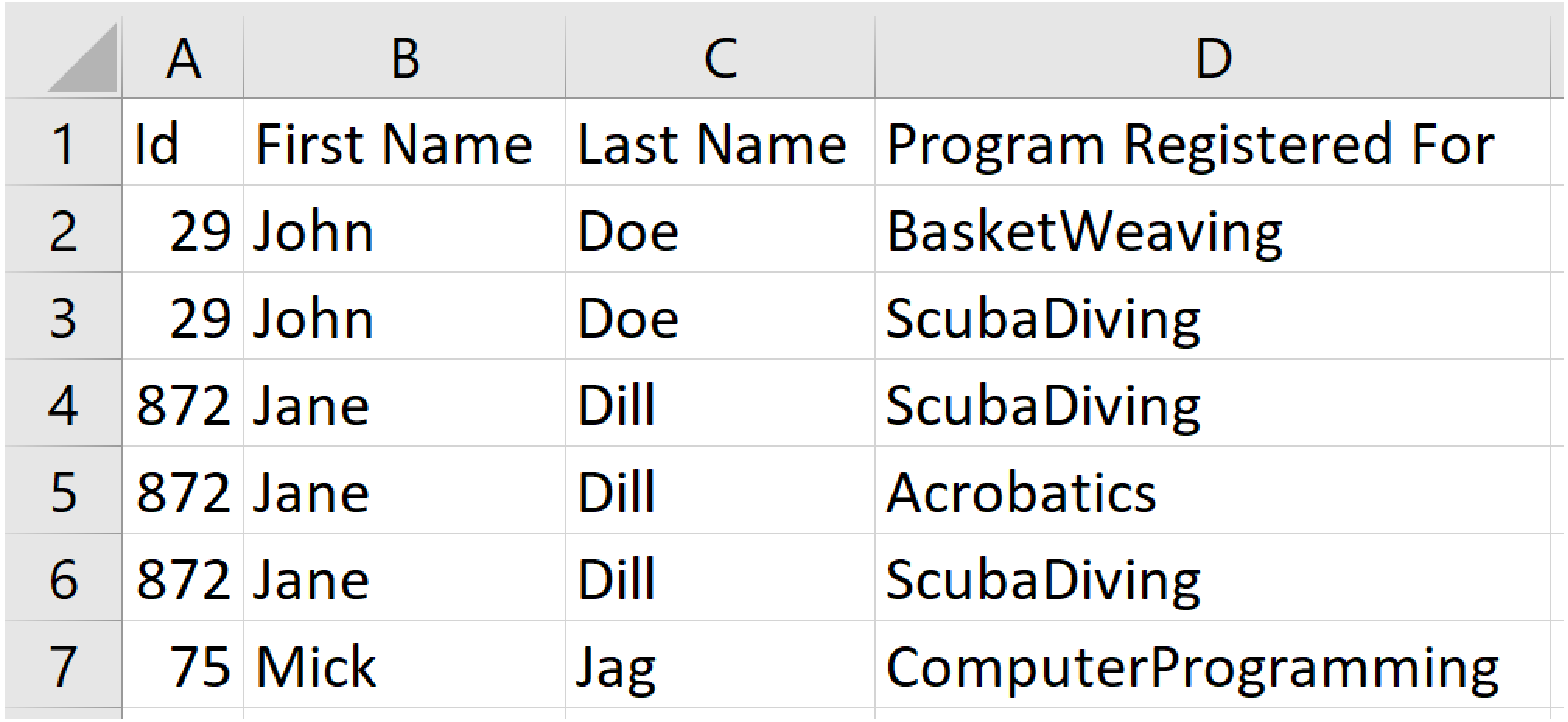
- 9 rows, 5 columns (people & **DOB & address**)
- Contacts from "Data Source **#3**"



	A	B	C	D	E
1	Id	First	Last	D.O.B.	Address
2	69435	Salli	Broxup	12/3/1991	305 Grover Lane, Sunny, AK
3	67121	Quintina	Lean	10/14/1963	305 Grover Lane, Sunny, AK
4	49617	Corny	Noller	12/13/1990	305 Grover Lane, Sunny, AK
5	86605	Yuri	Dalton	11/12/1980	800 Golden Leaf Street, Snowy, NM
6	22276	Doretta	Herche	9/21/2010	800 Golden Leaf Street, Snowy, NM
7	64465	Mata	Pierrepont	8/19/1970	800 Golden Leaf Street, Snowy, NM
8	32443	Othelia	Eastbury	8/4/1955	87834 Lyons Terrace, Rainy, OR
9	22082	Pansy	Mallya	8/4/1955	87834 Lyons Terrace, Rainy, OR
10	67526	Kata	Windus	10/4/1991	98 Paget Trail, Cloudy, WY

sample4.csv

- 6 rows, 4 columns (people & **each course registered for**)
- Course Registration transactions from "Data Source **#4**"



	A	B	C	D
1	Id	First Name	Last Name	Program Registered For
2	29	John	Doe	BasketWeaving
3	29	John	Doe	ScubaDiving
4	872	Jane	Dill	ScubaDiving
5	872	Jane	Dill	Acrobatics
6	872	Jane	Dill	ScubaDiving
7	75	Mick	Jag	ComputerProgramming

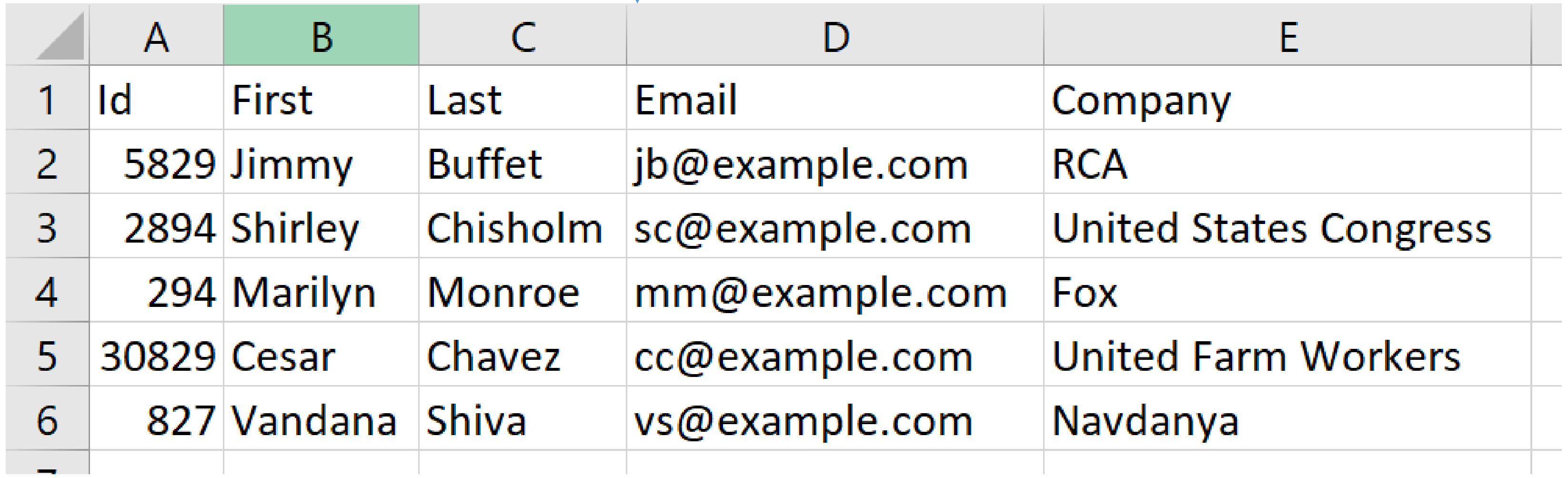
First 3 Lines Of Every Example (hidden in upcoming slides)

➢ import pandas	➢ “Please let me use the extra commands that come with ‘Pandas.’”
➢ pandas.set_option('expand_frame_repr', False)	➢ “Don’t do annoying line-wrapping when I ‘print()’ data that ‘Pandas’ has processed.”
➢ **df1 =** pandas.read_csv(**'c:\\yay\\sample1.csv'**)	➢ “Read ‘c:\yay\sample1.csv’ from my hard drive into Python. Save the Python copy into a variable/nickname called ‘**df1**.’” Notes: o I’ll use “**df2**” to import “**sample2.csv**,” etc. o I chose “df…” because Python calls the “**data type**” representing “2-D table-shaped data” a “Pandas **D**ata**F**rame.” o Online copies of examples might more inside “**.read_csv()**” to correctly handle dates, etc.

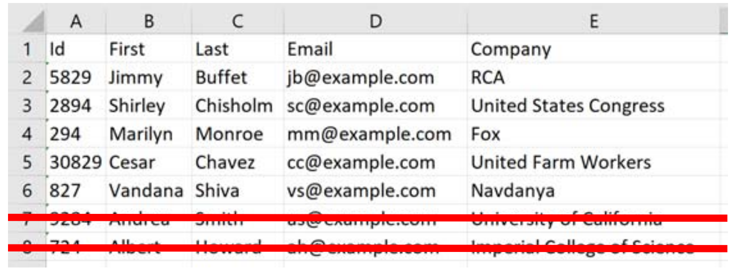
Example #1: CSV -> Pandas. Print. Export first five lines to new CSV.

```
print('---Here are all 7 lines---')
print(df1)
fivelinedf = df1.head(5)
fivelinedf.to_csv('C:\\yay\\out_fiveline.csv', index=False, quoting=1)
```

```
---Here are all 7 lines---
      Id    First      Last          Email                      Company
0   5829    Jimmy    Buffet  jb@example.com                          RCA
1   2894  Shirley  Chisholm  sc@example.com      United States Congress
2    294  Marilyn   Monroe   mm@example.com                          Fox
3  30829    Cesar    Chavez  cc@example.com          United Farm Workers
4    827  Vandana     Shiva  vs@example.com                     Navdanya
5   9284   Andrea     Smith  as@example.com     University of California
6    724   Albert    Howard  ah@example.com  Imperial College of Science
```



	A	B	C	D	E
1	Id	First	Last	Email	Company
2	5829	Jimmy	Buffet	jb@example.com	RCA
3	2894	Shirley	Chisholm	sc@example.com	United States Congress
4	294	Marilyn	Monroe	mm@example.com	Fox
5	30829	Cesar	Chavez	cc@example.com	United Farm Workers
6	827	Vandana	Shiva	vs@example.com	Navdanya

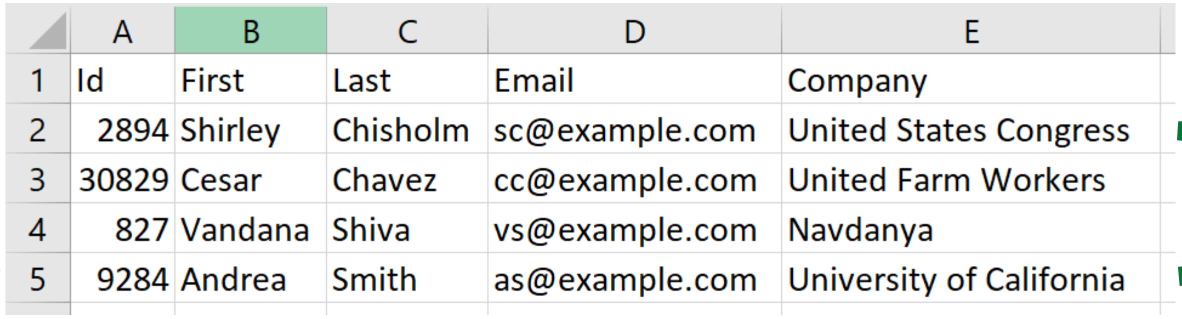


	A	B	C	D	E
1	Id	First	Last	Email	Company
2	5829	Jimmy	Buffet	jb@example.com	RCA
3	2894	Shirley	Chisholm	sc@example.com	United States Congress
4	294	Marilyn	Monroe	mm@example.com	Fox
5	30829	Cesar	Chavez	cc@example.com	United Farm Workers
6	827	Vandana	Shiva	vs@example.com	Navdanya
~~7~~	~~9284~~	~~Andrea~~	~~Smith~~	~~as@example.com~~	~~University of California~~
~~8~~	~~724~~	~~Albert~~	~~Howard~~	~~ah@example.com~~	~~Imperial College of Science~~

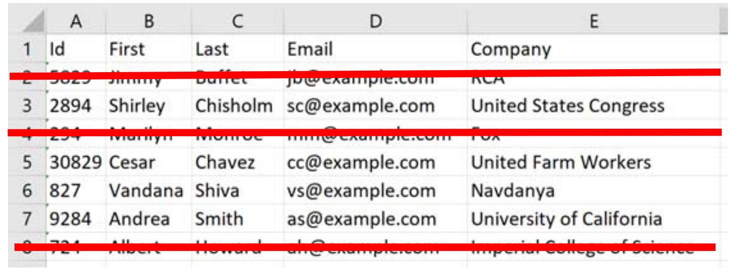
Example #2: Row Filtering

```
➢ print('---What is in "Last" for each row?---')
➢ lastNameSeries = df1['Last']
➢ print(lastNameSeries)

➢ print('---For each row, does "Last" start w/ "C" or "S"?---')
➢ lastCSBooleanSeries = lastNameSeries.str.startswith('C') | lastNameSeries.str.startswith('S')
➢ print(lastCSBooleanSeries)

➢ lastCSdf = df1[lastCSBooleanSeries]
➢ lastCSdf.to_csv('C:\\yay\\out_lastcs.csv', index=False, quoting=1)
```

```
---What is in "Last" for each row?---
0      Buffet
1    Chisholm
2      Monroe
3      Chavez
4       Shiva
5       Smith
6      Howard
Name: Last, dtype: object
---For each row, does "Last" start w/ "C" or "S"?---
0    False
1     True
2    False
3     True
4     True
5     True
6    False
Name: Last, dtype: bool
```



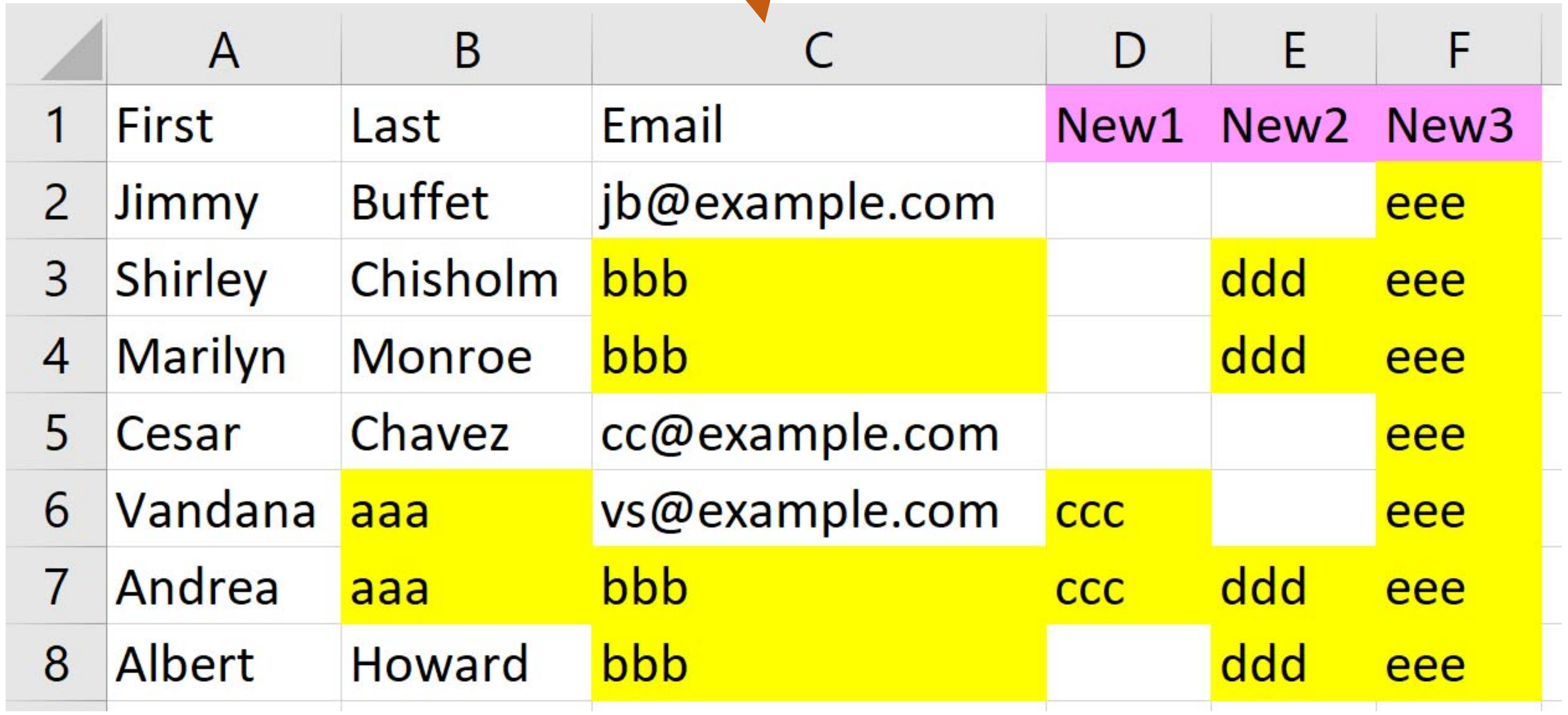
	A	B	C	D	E
1	Id	First	Last	Email	Company
2	2894	Shirley	Chisholm	sc@example.com	United States Congress
3	30829	Cesar	Chavez	cc@example.com	United Farm Workers
4	827	Vandana	Shiva	vs@example.com	Navdanya
5	9284	Andrea	Smith	as@example.com	University of California



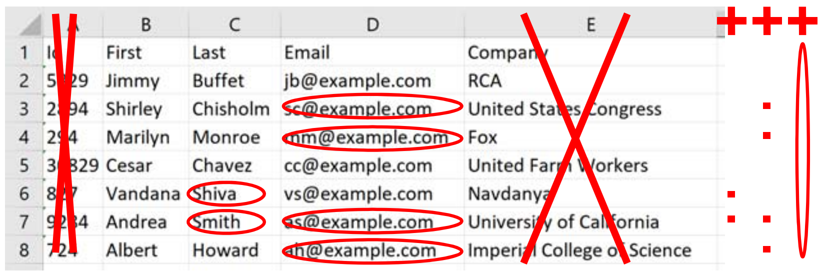
	A	B	C	D	E
1	Id	First	Last	Email	Company
~~2~~	~~5829~~	~~Jimmy~~	~~Buffet~~	~~jb@example.com~~	~~RCA~~
3	2894	Shirley	Chisholm	sc@example.com	United States Congress
~~4~~	~~294~~	~~Marilyn~~	~~Monroe~~	~~mm@example.com~~	~~Fox~~
5	30829	Cesar	Chavez	cc@example.com	United Farm Workers
6	827	Vandana	Shiva	vs@example.com	Navdanya
7	9284	Andrea	Smith	as@example.com	University of California
~~8~~	~~724~~	~~Albert~~	~~Howard~~	~~ah@example.com~~	~~Imperial College of Science~~

Example #3: Complex Cell Updates

```
➢ theseRowsLastNamesStartWithCapitalS = df1['Last'].str.startswith('S')
➢ theseRowsHaveA4InTheirId = df1['Id'].astype(str).str.contains('4')
➢ df1.loc[theseRowsLastNamesStartWithCapitalS,'Last'] = 'aaa'
➢ df1.loc[theseRowsHaveA4InTheirId,'Email'] = 'bbb'
➢ df1.loc[theseRowsLastNamesStartWithCapitalS,'New1'] = 'ccc'
➢ df1.loc[theseRowsHaveA4InTheirId,'New2'] = 'ddd'
➢ df1['New3'] = 'eee'
➢ df1 = df1.drop(['Id','Company'], axis=1)
➢ df1.to_csv('C:\\yay\\out_complexupdates.csv', index=False, quoting=1)
```



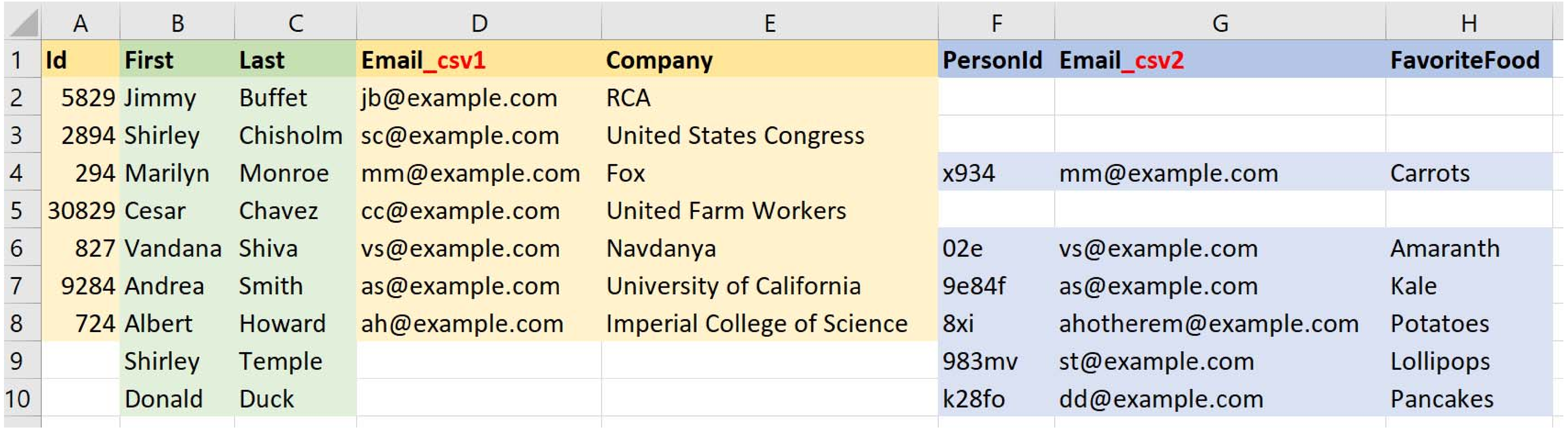
	A	B	C	D	E	F
1	First	Last	Email	New1	New2	New3
2	Jimmy	Buffet	jb@example.com			eee
3	Shirley	Chisholm	bbb		ddd	eee
4	Marilyn	Monroe	bbb		ddd	eee
5	Cesar	Chavez	cc@example.com			eee
6	Vandana	aaa	vs@example.com	ccc		eee
7	Andrea	aaa	bbb	ccc	ddd	eee
8	Albert	Howard	bbb		ddd	eee



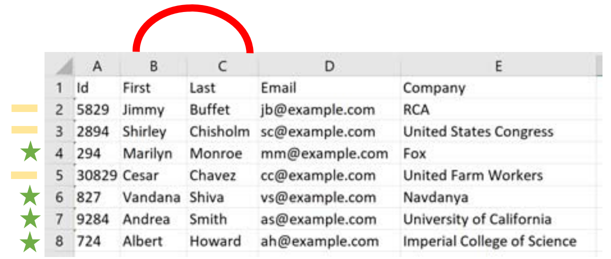
	A	B	C	D	E
1	~~Id~~	First	Last	Email	~~Company~~
2	~~5[illegible]29~~	Jimmy	Buffet	jb@example.com	~~RCA~~
3	~~2[illegible]94~~	Shirley	Chisholm	sc@example.com	~~United States Congress~~
4	~~2[illegible]4~~	Marilyn	Monroe	mm@example.com	~~Fox~~
5	~~3[illegible]829~~	Cesar	Chavez	cc@example.com	~~United Farm Workers~~
6	~~8[illegible]7~~	Vandana	Shiva	vs@example.com	~~Navdanya~~
7	~~9[illegible]34~~	Andrea	Smith	as@example.com	~~University of California~~
8	~~7[illegible]4~~	Albert	Howard	ah@example.com	~~Imperial College of Science~~

Example #4: Multi-Column VLOOKUP

```
➢ betterdf2 = df2.rename(columns = {'LastName':'Last', 'FirstName':'First', 'Em':'Email'})
➢ outermergedf = df1.merge(betterdf2, how='outer', on=['Last', 'First'], suffixes=('_csv1', '_csv2'))
➢ outermergedf.to_csv('C:\\yay\\out_outermerge.csv', index=False, quoting=1)
```

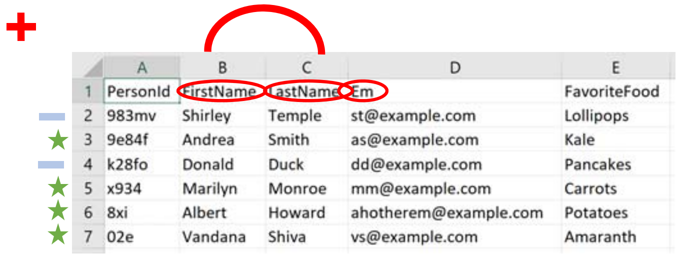


	A	B	C	D	E	F	G	H
1	Id	First	Last	Email_csv1	Company	PersonId	Email_csv2	FavoriteFood
2	5829	Jimmy	Buffet	jb@example.com	RCA			
3	2894	Shirley	Chisholm	sc@example.com	United States Congress			
4	294	Marilyn	Monroe	mm@example.com	Fox	x934	mm@example.com	Carrots
5	30829	Cesar	Chavez	cc@example.com	United Farm Workers			
6	827	Vandana	Shiva	vs@example.com	Navdanya	02e	vs@example.com	Amaranth
7	9284	Andrea	Smith	as@example.com	University of California	9e84f	as@example.com	Kale
8	724	Albert	Howard	ah@example.com	Imperial College of Science	8xi	ahotherem@example.com	Potatoes
9		Shirley	Temple			983mv	st@example.com	Lollipops
10		Donald	Duck			k28fo	dd@example.com	Pancakes



		A	B	C	D	E
	1	Id	First	Last	Email	Company
▬	2	5829	Jimmy	Buffet	jb@example.com	RCA
▬	3	2894	Shirley	Chisholm	sc@example.com	United States Congress
★	4	294	Marilyn	Monroe	mm@example.com	Fox
▬	5	30829	Cesar	Chavez	cc@example.com	United Farm Workers
★	6	827	Vandana	Shiva	vs@example.com	Navdanya
★	7	9284	Andrea	Smith	as@example.com	University of California
★	8	724	Albert	Howard	ah@example.com	Imperial College of Science

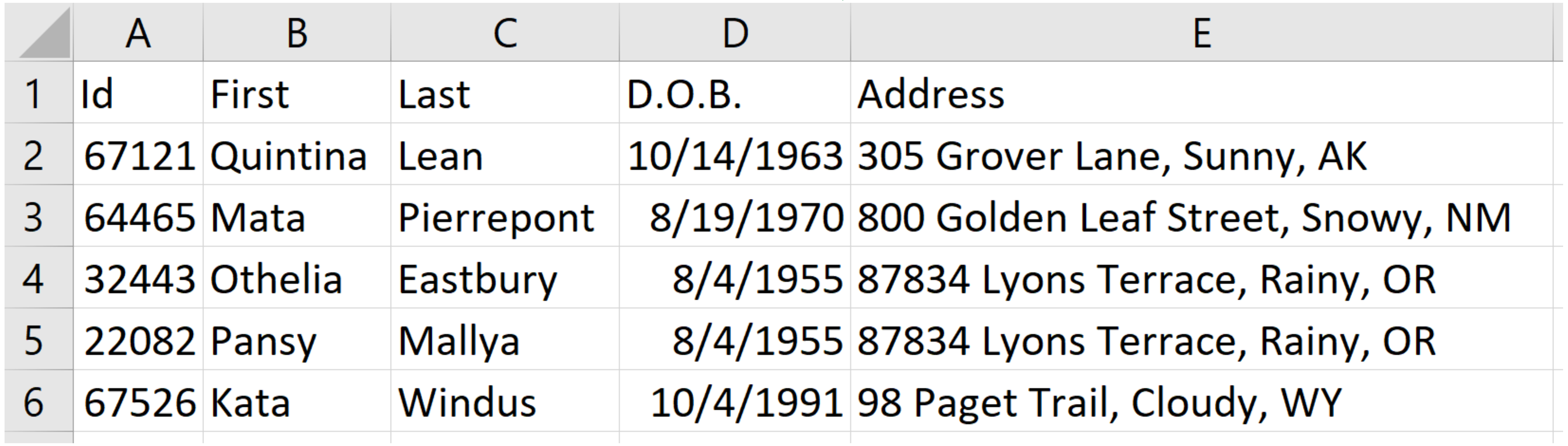
+



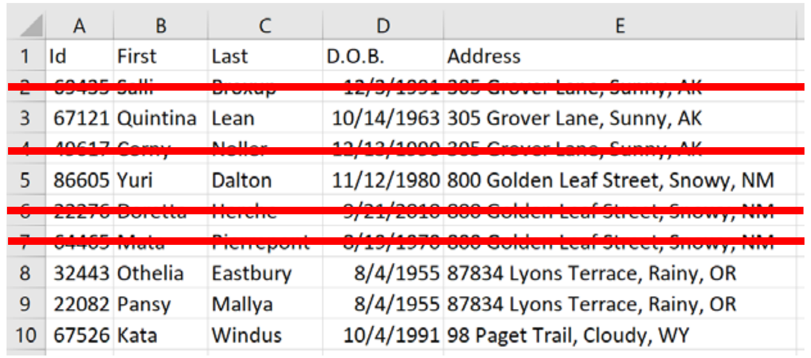
		A	B	C	D	E
	1	PersonId	FirstName	LastName	Em	FavoriteFood
▬	2	983mv	Shirley	Temple	st@example.com	Lollipops
★	3	9e84f	Andrea	Smith	as@example.com	Kale
▬	4	k28fo	Donald	Duck	dd@example.com	Pancakes
★	5	x934	Marilyn	Monroe	mm@example.com	Carrots
★	6	8xi	Albert	Howard	ahotherem@example.com	Potatoes
★	7	02e	Vandana	Shiva	vs@example.com	Amaranth

Example #5: Filtering on Aggregations

```
➢ groupingByAddress = df3.groupby('Address')
➢ groupedDataFrame = groupingByAddress.apply(lambda x: x[x['D.O.B.'] == x['D.O.B.'].min()])
➢ outputdf = groupedDataFrame.reset_index(drop=True)
➢ outputdf.to_csv('C:\\yay\\out_oldest_person_per_address.csv', index=False, quoting=1)
```



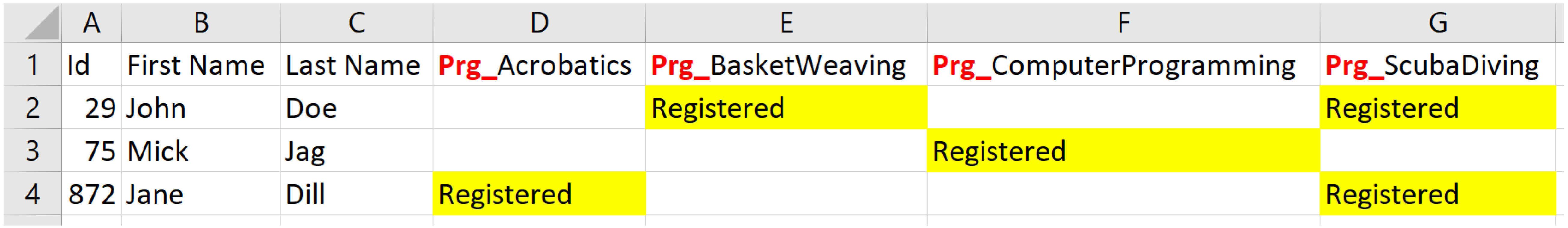
	A	B	C	D	E
1	Id	First	Last	D.O.B.	Address
2	67121	Quintina	Lean	10/14/1963	305 Grover Lane, Sunny, AK
3	64465	Mata	Pierrepont	8/19/1970	800 Golden Leaf Street, Snowy, NM
4	32443	Othelia	Eastbury	8/4/1955	87834 Lyons Terrace, Rainy, OR
5	22082	Pansy	Mallya	8/4/1955	87834 Lyons Terrace, Rainy, OR
6	67526	Kata	Windus	10/4/1991	98 Paget Trail, Cloudy, WY



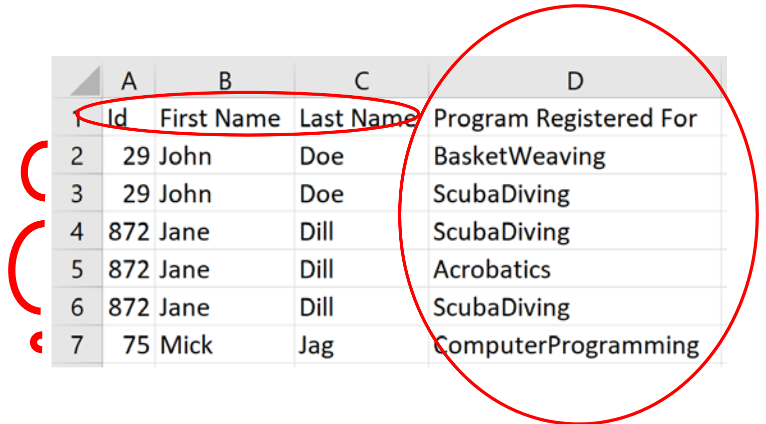
	A	B	C	D	E
1	Id	First	Last	D.O.B.	Address
~~2~~	~~69435~~	~~Salli~~	~~Broxup~~	~~12/3/1991~~	~~305 Grover Lane, Sunny, AK~~
3	67121	Quintina	Lean	10/14/1963	305 Grover Lane, Sunny, AK
~~4~~	~~40617~~	~~Corny~~	~~Noller~~	~~12/13/1990~~	~~305 Grover Lane, Sunny, AK~~
5	86605	Yuri	Dalton	11/12/1980	800 Golden Leaf Street, Snowy, NM
~~6~~	~~22276~~	~~Doretta~~	~~Herche~~	~~9/21/2010~~	~~800 Golden Leaf Street, Snowy, NM~~
~~7~~	~~64465~~	~~Mata~~	~~Pierrepont~~	~~8/19/1970~~	~~800 Golden Leaf Street, Snowy, NM~~
8	32443	Othelia	Eastbury	8/4/1955	87834 Lyons Terrace, Rainy, OR
9	22082	Pansy	Mallya	8/4/1955	87834 Lyons Terrace, Rainy, OR
10	67526	Kata	Windus	10/4/1991	98 Paget Trail, Cloudy, WY

Example #6: Pivoting log -> people

```
➢ import numpy
➢ df4['Program Registered For'] = 'Prg_' + df4['Program Registered For']
➢ non_program_columns = list(filter(lambda x: x != 'Program Registered For', df4.keys()))
➢ pivotdf = pandas.pivot_table(df4, index=non_program_columns, columns='Program Registered For', aggfunc=numpy.size)
➢ pivotdf[pandas.notnull(pivotdf)] = 'Registered'
➢ pivotdf.reset_index(inplace=True)
➢ pivotdf.to_csv('C:\\yay\\out_pivoted_program_registrations.csv', index=False, quoting=1)
```



	A	B	C	D	E	F	G
1	Id	First Name	Last Name	Prg_Acrobatics	Prg_BasketWeaving	Prg_ComputerProgramming	Prg_ScubaDiving
2	29	John	Doe		Registered		Registered
3	75	Mick	Jag			Registered	
4	872	Jane	Dill	Registered			Registered



	A	B	C	D
1	Id	First Name	Last Name	Program Registered For
2	29	John	Doe	BasketWeaving
3	29	John	Doe	ScubaDiving
4	872	Jane	Dill	ScubaDiving
5	872	Jane	Dill	Acrobatics
6	872	Jane	Dill	ScubaDiving
7	75	Mick	Jag	ComputerProgramming

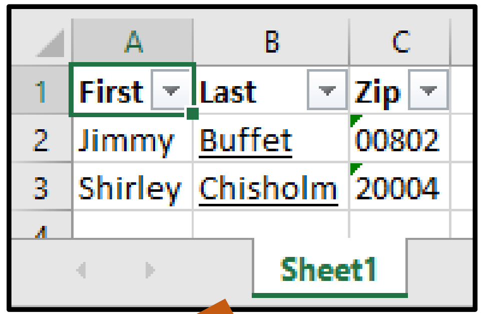
Pro Tip: Close Excel

- If your Python program crashes when it gets to **“.to_csv(…)**”
  - Is the CSV you’re trying to save open in Excel?
  - Close Excel and run your program again

Bonus: Excel files

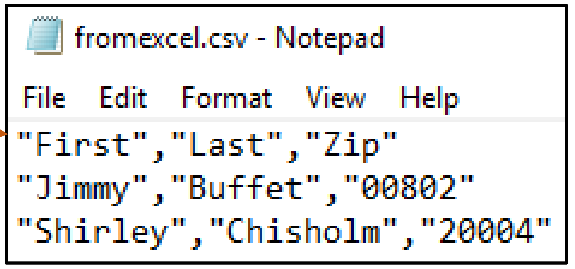
If your “IDE” includes a new-ish version of Python & Pandas, plus plugins like “xlrd”…

- **Import:**
  - myNickname = pandas.**read_excel**(…)
    - Works w/ simple, starts-in-A1 Excel tables
    - Avoids XLS→CSV headache *(Excel XLS→CSV loves to strip your leading 0’s.)* 😨
- **Export:**   myNickname.**to_excel**(…)
  - myNickname.**to_excel**(…)



	A	B	C
1	First	Last	Zip
2	Jimmy	Buffet	00802
3	Shirley	Chisholm	20004
4			

- XLS→CSV with Python:
  - dfx = pandas.**read_excel**('C:\\yay\\fromexcel.xlsx', 'Sheet1', **converters**={'Zip':**str**})
  - dfx.**to_csv**('C:\\yay\\fromexcel.csv', index=False, quoting=1)



```
"First","Last","Zip"
"Jimmy","Buffet","00802"
"Shirley","Chisholm","20004"
```

Recap

Desired Takeaways

- "I saw words today that … looked relevant …"

  (**.min()**, **'Email'**, **.to_csv()** …)

- "That code is **way** easier to 'sight read' than Excel VBA."

- "Wow, that's a lot of action for so little code."

And to make my day…

- "I'm pretty handy copying, pasting, and modifying fancy Excel formulas I find online.
  I think I could figure out how to do the same with this."

Further Resources

- **Today's slides** with **code editable/runnable online & quizzes!** + "common operations & how to use them" list: *https://tinyurl.com/**pypancsv***
- **Hands-On Trainings**: *https://tinyurl.com/**handson-**pypancsv*
- **IDEs**:
  - **WinPython** (desktop) -- no admin rights needed *https://tinyurl.com/PyPanCsvWinIde*
  - **CodeBunk / Repl.it** (online) -- NEVER use private data! *https://codebunk.com/b/* & *https://repl.it/languages/python3*
- **Practical Business Python** blog *(start @ end & skim to now)*: *http://pbpython.com*

Here's a cute picture of Pandas

Questions? Revisit examples?

Hands-on training mailing list:

https://tinyurl.com/handson-pypancsv

Runnable code / slides / exercises: https://tinyurl.com/pypancsv