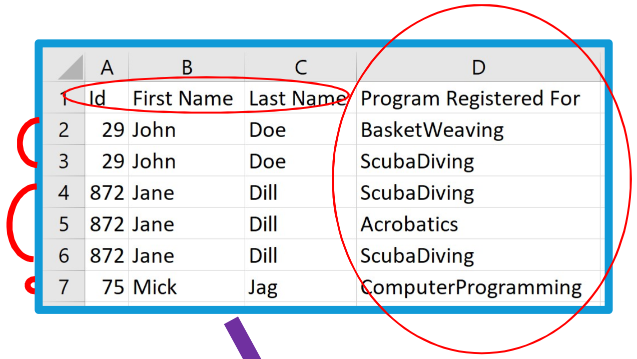
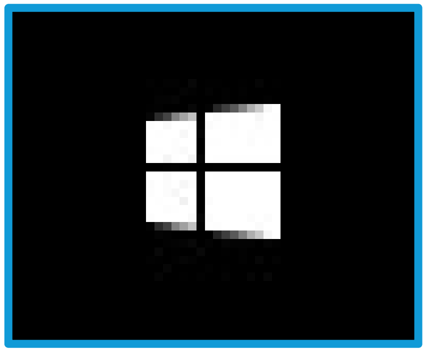
Python for Spreadsheet Manipulation 101

Twin Cities Salesforce Saturdays

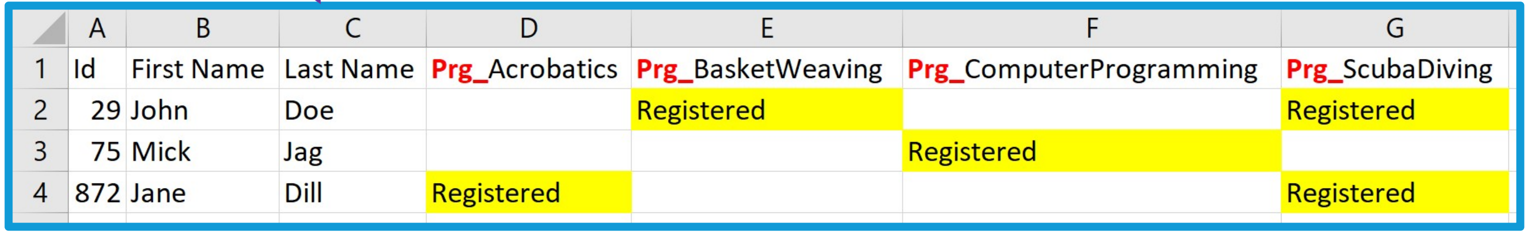
© "pypancsv" 2018



	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



How I Found Python



	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

Excel vs/and Python

(Today, just Python, for practice!)

Syllabus

101 (today)

- “Hello, World!”
- Programmer-speak
- Sample Code!
    - Import/Export CSV/XLSX
    - Counting & displaying things
    - Duplicates & uniques
    - Sorting rows
    - Adding/dropping columns
    - Date-time gotchas
    - Selectively editing cells
    - Basic matching / VLOOKUP

🌊🌊 Yes, it’s a lot! 🌊🌊

1. Watch me EXPLAIN so:
   - “Cool! 🌟😍❤”
   - Recognition when Googling
2. 👍 – keep up:
   - To reinforce the experience

102 (future)

- Anything we miss today
- More matching/VLOOKUP problems
- More “hard problems”
- BYO problem & sample data; let’s solve it!

Links

- Every link will start with https://link.stthomas.edu/sfpy201810- ...
- Struggling to type fast enough? Code snippets at:
  - [https://link.stthomas.edu/sfpy201810-**info**](https://link.stthomas.edu/sfpy201810-info)
    - (Online folks: you're already here – it's where you got the webinar link.)
  - "Hands-on" slides will indicate which exercise from this "info sheet" we're on using an orange cloud with a number in it!

3F

Let’s Run A Program

https://link.stthomas.edu/sfpy201810-hello

https://link.stthomas.edu/sfpy201810-hello

1A

- Running Codebunk examples (“fork” + don’t log in)
- Any problems running it?
    - (Remind me to check the chat)

https://link.stthomas.edu/sfpy201810-hello

1B

- Change `Hello World` to `Yay Us` and run your code.
- Any problems? Does “Yay Us” show up?

Code Fragment Jargon & IDEs

- **Expression:** code that *is* a value.
  Like a single Excel cell's formula.
  - `'Hello World'`
  - `'Yay Us'`
  - `type('Hello World')`
  - `(1 + 1) / 5`
  - `'Amanda'.startsWith('Z')`
- **Statement:** code that *does*.
  Smallest runnable program.
  Statement : Program :: Sentence : Essay
  - `print(SOME EXPRESSION HERE)`
  - `cool_variable_Name = SOME EXPRESSION HERE`
- **Operat(-ion/-or) / Function / Method:**
  expression glue *(→expression or →statement)*
  - `+`
  - `print(…)`
  - `type(…)`
- **Comments:** code fragment for humans
  - `# One-line comment`

  -
    ```
    '''
    Multi-line comment:
    For really long comments!
    '''
    ```
- **IDE:** text editor with a "Run" button
  - Install & run on a computer you control for corporate data

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)

Why “Dimension” Matters

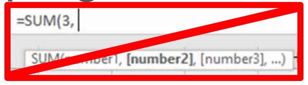
- “Dimension” & “Kind” work together to constrain what “**operations**” we can do to data. Can we …

+, - ?	0D #; 0D text if “+” means “concatenate”
fetch 1st letter?	0D text data
<, == ?	0D number, 0D text …
SELECT? • fetch “item #3” *(2D→1D; 1D→0D)* • fetch “odd-numbered” items *(2D→2D; 1D→1D)*	1D & 2D data
ITERATE? *(inspect each item, potentially altering its value)* • multiply each by 3 • all-caps any item that starts with a P	1D & 2D data
AGGREGATE? *(combine all the items together into just one value)* • max • sum	1D & 2D data

♥ Data Types = Easier "Expression" Writing

- Tricky #1: Fewer hints about "expression operations <u>while</u> you program

  (in online manuals, though)



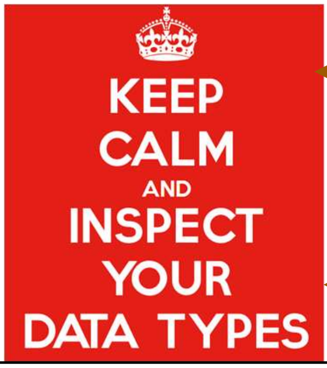
=SUM(3,

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

- Tricky #2: Not just "*around*" & "*between*" operations like Excel's `ISNUMBER("apple")` & `1+4`
  - Also "*after*" operations, connected by a period, like `"Banana".lower()`
  - Worse: "*after*" operations in Pandas w/ random extra period, like `….str.lower()`
  - Or: "after" operations in Pandas that launch straight into brackets, like `ExpressionHere[…]`

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(…))` w/ `9 - 4`, `5`, `2`, `5 < 2`, `False`, etc.
- Copy/paste smaller problems back together, just like you do with big Excel formulas

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”** (notes to self)

No shame in “programming by Google”

print command in python

Let’s Try That!

- Stay on your current “code bunk.”
  Already close it? Re-“fork” https://link.stthomas.edu/sfpy201810-hello
- Backspace or “comment out” your old code
  (Who can guess how we “comment out” code?)

Type and run, one at a time. Surprises?

1C

- `print('Hello World')`
- `print(type('Hello World'))`
- `print(5)`
- `print(type(5))`
- `print(None)`
- `print(type(None))`
- `print(False)`
- `print(type(False))`
- `print(3 * 2.5 * 4)`
- `print(type(3 * 2.5 * 4))`
- `print(3 * 2.5 * 4 < 1)`
- `print(type(3 * 2.5 * 4 < 1))`
- `myFirstVariable = 3 * 2.5 * 4`
- `print(myFirstVariable)`
- `print(type(myFirstVariable))`
- `print(myFirstVariable < 1)`
- `print(type(myFirstVariable < 1))`
- `print('Bye!')`

- Boldface: The “outermost expression” within the “`print(...)`” operator
- Underline: The expression we’re interested in seeing the “value” or the “data type” of

Answer Key. Surprises? (Chat room, surprises?)

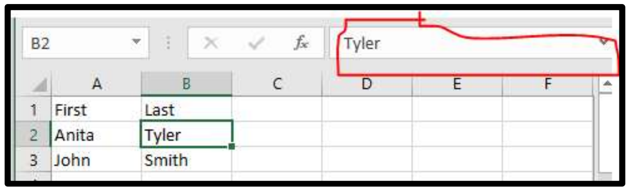
print(**'Hello World'**)	Hello World
print(**type('Hello World')**)	<class 'str'>
print(**5**)	5
print(**type(5)**)	<class 'int'>
print(**None**)	None
print(**type(None)**)	<class 'NoneType'>
print(**False**)	False
print(**type(False)**)	<class 'bool'>
print(**3 * 2.5 * 4**)	30.0
print(**type(3 * 2.5 * 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!

Expression-Nesting Pop Quiz

- "Angela".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				

Questions? (Chat room?)

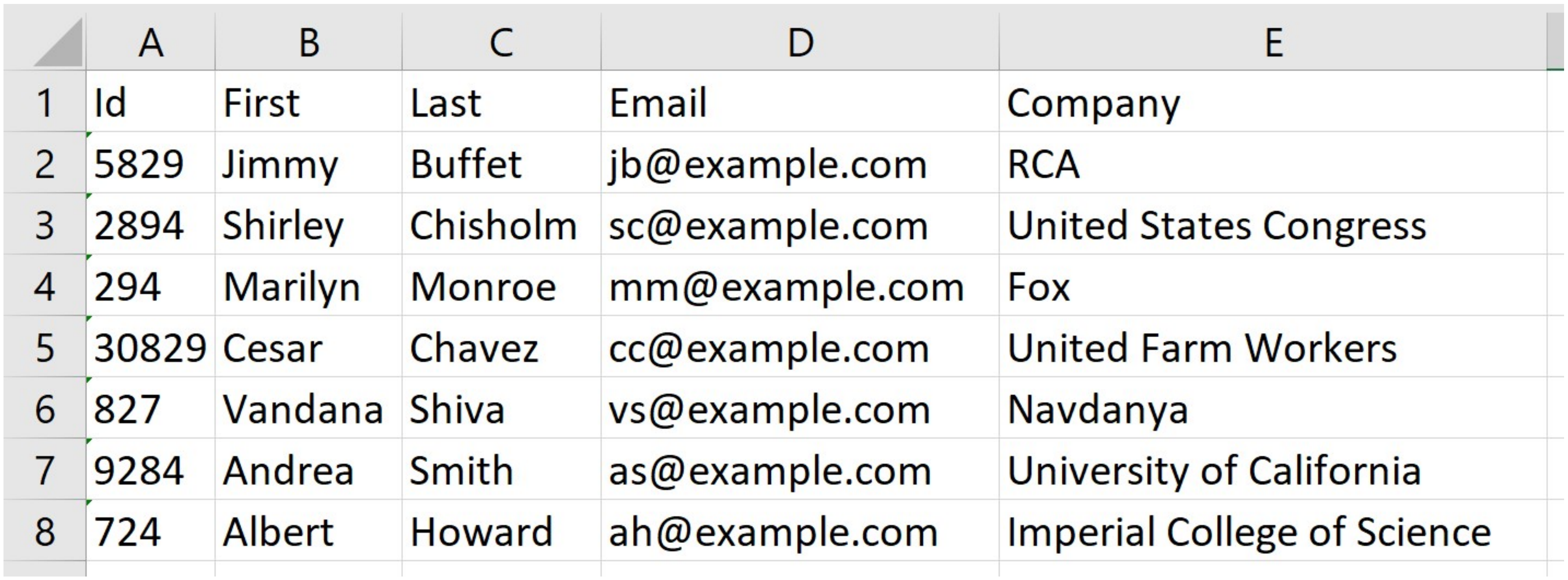
(Trouble getting code to run?)

Let’s look at a CSV file using Python

https://link.stthomas.edu/sfpy201810-readcsv

sample1.csv

- 7 rows, 5 columns (people & **employer**)



	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

https://link.stthomas.edu/sfpy201810-readcsv

2A

- 🍴 *(remember to “fork” it if it won’t run!)* 🍴
- Any problems running it?
    - (Remind me to check the chat)

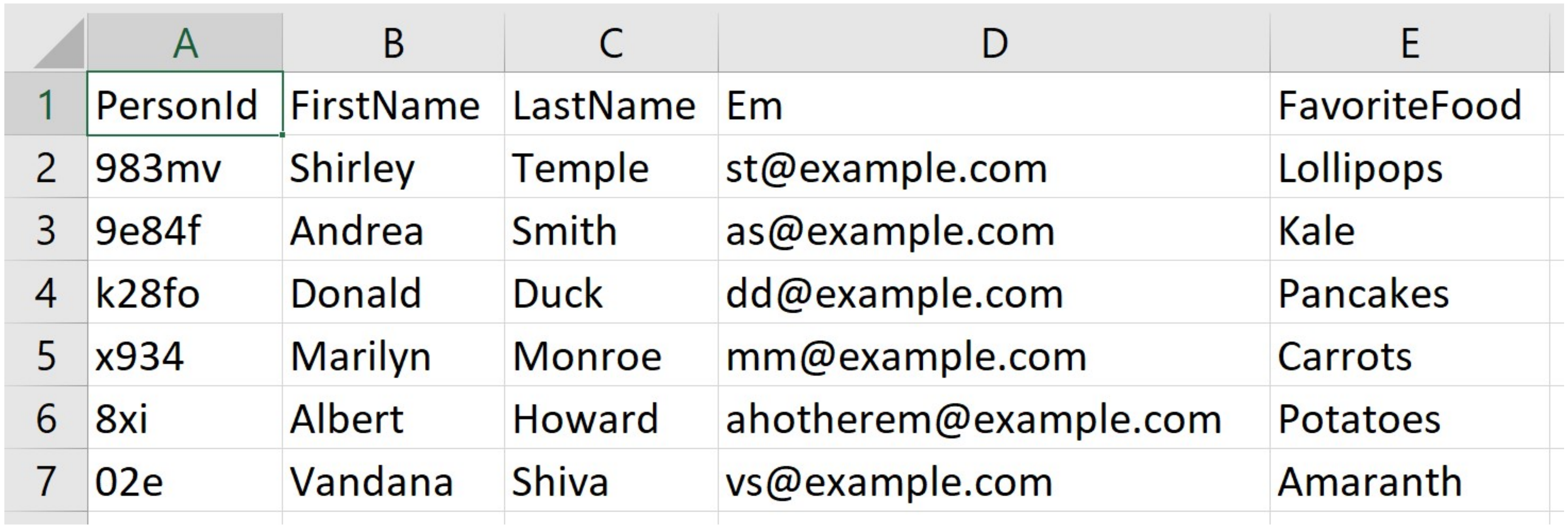
Excel, too!

- In addition to **`pandas.read_csv(…)`**, there's also **`pandas.read_excel(…)`**

- When we finish crafting a Pandas "DataFrame" that we like and saving it into a variable called, say, "**`outputdf`**," we can do:
    - **`outputdf.to_csv(…)`**
    - **`outputdf.to_excel(…)`**

sample2.csv

- 6 rows, 5 columns (people & **favorite food**)



	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

2B

https://link.stthomas.edu/sfpy201810-readcsv - **Edit the Code (CSV 2)**

Change all but the first occurrence of **`df1`** to **`df2`** and re-run.

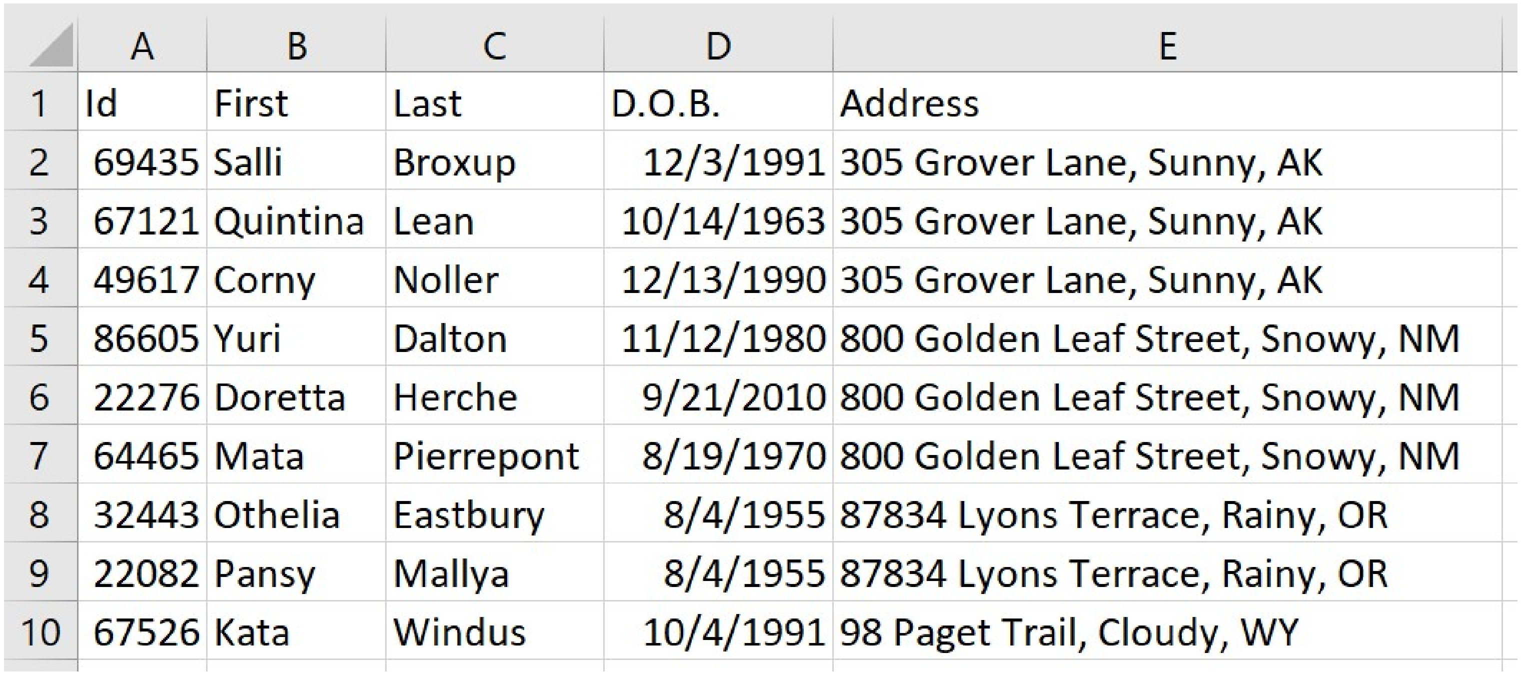
- There are 19 changes to make *(the last 19 lines of the program)*
- Don’t change the one at the very top that starts with **`df1 =`**

Review:

- Are you seeing people and their favorite foods?
- Is the total row count down from 7 to 6?
- Any problems? Questions?
  - (Remind me to check the chat)

sample3.csv

- 9 rows, 5 columns (people & **DOB** & address)



	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

2C

https://link.stthomas.edu/sfpy201810-readcsv - **Edit the Code (CSV 3)**

Change all but the first occurrence of `df2` to `df3` and re-run.

- There are 19 changes to make *(the last 19 lines of the program)*
- Don't change the one at the very top that starts with `df2 =`

Review:

- Are you seeing people and their addresses?
- Is the total row count up from 6 to 9?
- Any problems? Questions?
    - (Remind me to check the chat)

10-Minute Break

Let’s get a little bit harder

3 exercises, same code base.

(If you closed it, re-visit
https://link.stthomas.edu/sfpy201810-readcsv
and “fork” it as soon as the page loads.)

https://link.stthomas.edu/sfpy201810-readcsv - Edit the Code

2D

1. "Comment out" the whole `print(…)` section of code – all 19 lines.
    - Do this by adding 3 single quotes in a row, `'''`, both before and after that section of code.
2. At the end of the program, add the following new line of code:

```
print(df3['Address'].unique())
```

3. Run the code

Review:

- Do you see the following output?
    - **['305 Grover Lane, Sunny, AK' '800 Golden Leaf Street, Snowy, NM' '87834 Lyons Terrace, Rainy, OR' '98 Paget Trail, Cloudy, WY']**
- Any problems? Questions?
    - (Remind me to check the chat)

https://link.stthomas.edu/sfpy201810-readcsv - **Edit the Code**

2E

1. At the end of the program, add the following new line of code:

```
print(len(df3['Address'].unique()))
```

   - Tip: It's like the line before it, only with `len(…)` inside the `print(…)`
2. Run the code

Review:

- Do you see the following output right below your list of addresses?
  - `4`
- Any problems? Questions?
  - (Remind me to check the chat)

Useful things to wrap around `… .unique(…)`

- `len(…)` – gives you a count of the unique values
- `list(…)` – useful when you want to copy/paste the output to your clipboard with commas between the values
- `sorted(…)` – the same output as `list(…)`, only case-sensitive alpha order
- `sorted(…, key=str.lower)` – the same output as `sorted(…)`, only case-**in**sensitive alpha order

https://link.stthomas.edu/sfpy201810-readcsv - **Edit the Code**

2F

1. “Comment out” the last 2 lines of code you added.
   - Do this by putting a **#** at the beginning of each of the 2 lines of code.
2. At the end of the program, add the following 3 new lines of code:

```
print(df3.drop_duplicates(['Address'], keep=False))
print(len(df3.drop_duplicates(['Address'], keep=False)))
print(len(df3.drop_duplicates(['Address','D.O.B.'], keep=False)))
```

3. Run the code

Review:

- Do you see…
  - First, **the contents of “row 8”** *(really 9)*, which is the only person who lives alone in CSV 3?
  - Second, the number **1** *(the # of rows w/ a unique address)*?
  - Third, the number **7** *(the # of rows w/ a unique address+DOB combo—i.e. no roomie twins)*?
- Any problems? Questions?
  - (Remind me to check the chat)

Note `df3` was “reusable”

- `.drop_duplicates(…)` didn’t really do anything to df3 in that code
  - We were just `print(…)` -ing copies
- Know that there ARE ways to alter the contents of our variable “`df3`”

We did some neat stuff

- Read a CSV file off the world wide web into our program
- Displayed a copy of that CSV file on the screen in all its ugly glory
- Displayed just its “head” and “tail” *(handy if it’s 3,000 lines long)*
- Counted lots of things with “`len(…)`”
- Combined “`len(…)`” & “`['Address'].unique()`” into an address count
- Displayed just “unique-data” rows with “`.drop_duplicates(…)`”
    - `.drop_duplicates(…)` has other settings that allow us to leave in the first or last of any duplicate rows, instead of suppressing all rows that have a duplicate.

If you can display a DataFrame, you can export it**

- If `print(type(…))` displays **<class 'pandas.core.frame.DataFrame'>**, "…" is an **expression** to which you can append `.to_csv(…)` or `.to_excel(…)`
  - Pro tip: the `.to_csv(…)` command gets lengthy.
    Save your "DataFrame" to a "variable" as in the example below.

```
outputdf = df3.drop_duplicates(['Address'], keep=False)
outputdf.to_csv('C:\\myfolder\\myfile.csv', index=False, quoting=1)
```

*** Not today. You're running code online.*

More Theory:
DataFrames, Series, & Lists – Oh My!

DataFrames & Series & Lists – oh my!

Data Type	Comes With	Dimension
DataFrame	Pandas plugin	2-D (Table-shaped)
Series	Pandas plugin	1-D (List-shaped)
List	Python by default	1-D (List-shaped)

1-D AND 2-D data actions

- Select sub-members
  - (1-D: “select cells” / “select items”)
  - (2-D: “select columns”)
  - (2-D: “select rows”)

- “Sort” the data
  - (1-D: plain-old sort)
  - (2-D: sort entire “rows” after specifying “columns” whose cell contents will be used for sorting)

2-D data actions

- Adding a “column”
- Dropping a “column”
- Re-ordering “columns”
- Renaming a “column label”
- Importing from a spreadsheet file
- Exporting to a spreadsheet file

1-D data actions

- Editing the contents of cells/items based on other “**corresponding**” 1-D data

- Using “0-D”-specific operations on the contents of cells/items
  - `.isin(…)`
  - `.notnull()`
  - `.str.startwith(…)`
  - `<`
  - `+`

So?

- Programming is “working blind” compared to Excel.
- When stuck, helpful to “think like a computer” about what you’re “really trying to do to your data.”

Pandas Index

Pandas Index

- Pandas does a lot of its **“inter-column” / “corresponding data” magic** based on **row numbering**, which it calls “indexing.”

- **Usually safe to think of “indexes” as a “row number” or “row ID”**
  - “Row ID” probably best. Get used to seeing:
    - “Missing” row IDs (0, 2, 6, 7)
    - “Out-of-order” row IDs (3, 0, 1, 2)

  - Know that more complicated “indexes” exist
    - “Named” row IDs ('983mv', '9e84f', 'k28fo', 'x934', '8xi', '02e‘)
    - Multi-level indexes (when doing advanced pivoting & grouping)
    - Technically, column names are also indexes

Visual Cues

DataFrames vs. Series vs. Lists

What “DataFrames” look like

- `print(type(ExpressionHere))` displays **<class 'pandas.core.frame.DataFrame'>**
- `print(ExpressionHere)` looks something like:

With generic row IDs

```
   FirstName LastName PersonId
0    Shirley   Temple   983mv
1     Andrea    Smith   9e84f
2     Donald     Duck   k28fo
3    Marilyn   Monroe    x934
4     Albert   Howard     8xi
5    Vandana    Shiva     02e
```

With “named” row IDs

```
          FirstName LastName
PersonId
983mv       Shirley   Temple
9e84f        Andrea    Smith
k28fo        Donald     Duck
x934        Marilyn   Monroe
8xi          Albert   Howard
02e         Vandana    Shiva
```

With generic row IDs, sorted by LastName

```
   FirstName LastName PersonId
2     Donald     Duck    k28fo
4     Albert   Howard      8xi
3    Marilyn   Monroe     x934
5    Vandana    Shiva      02e
1     Andrea    Smith    9e84f
0    Shirley   Temple    983mv
```

- Nothing displayed below the last row
- Multiple data columns allowed. Inherently ***2-DIMENSIONAL***.
- “Data column” labels as high as they can go, right-aligned over data
- “Row IDs” at far left. 1) no label or 2) “lowered & left-aligned” if “named”
  - *(“Named” happens when you use a special command to convert a data column into a “row ID”)*

What “Series” look like

- `print(type(ExpressionHere))` displays **<class 'pandas.core.series.Series'>**
- `print(ExpressionHere)` looks something like:

“First Name” column	**“First Name” column *(with “named” row IDs)***	**Column w/ “Does this ‘First Name’ cell start with ‘A’?”**	**Row “2”**
	`PersonId`		
`0     Shirley`	`983mv     Shirley`	`0     False`	
`1      Andrea`	`9e84f      Andrea`	`1      True`	`PersonId                k28fo`
`2      Donald`	`k28fo      Donald`	`2     False`	`FirstName              Donald`
`3     Marilyn`	`x934      Marilyn`	`3     False`	`LastName                 Duck`
`4      Albert`	`8xi        Albert`	`4      True`	`Em            dd@example.com`
`5     Vandana`	`02e       Vandana`	`5     False`	`FavoriteFood        Pancakes`
`Name: FirstName, dtype: object`	`Name: FirstName, dtype: object`	`Name: FirstName, dtype: bool`	`Name: 2, dtype: object`

- “Name” *(if applicable)* & “**Data Type** of contents” displayed below last row
- Only 1 “data” column allowed. Inherently ***1-DIMENSIONAL***.
  - *(“PersonId as ‘row number’” or “Column label as ‘row number’” don’t count as a “column” – they’re the “index”)*
- No label at top for “data” column
- “Row IDs” still at far left. 1) no label or 2) “lowered & left-aligned” if “named”
  - *(“Named” happens when you use a special command to convert a data column into a “row ID”)*

What “Lists” look like

- `print(type(ExpressionHere))` displays **<class '<u>list</u>'>**
- `print(ExpressionHere)` looks something like:
  - `['Shirley', 'Andrea', 'Donald', 'Marilyn', 'Albert', 'Vandana']`
  - `[False, True, False, False, True, False]`
  - `['k28fo', 'Donald', 'Duck', 'dd@example.com', 'Pancakes']`
  - `['PersonId', 'FirstName', 'LastName', 'Em', 'FavoriteFood']`

- Single line. Inherently ***<u>1-DIMENSIONAL</u>***.
  - *(Yes, the “1-dimensional” bit means they “play nicely” with Series and vice-versa!)*
- Comma-separated values
- Square brackets
- You can’t see it, but implied “item numbers” ALWAYS **<u>0</u>**, 1, 2, 3… **in order**.
  - *(The “implied item numbers” can be used to “select” certain items out of the list.)*

Hands-On

https://link.stthomas.edu/sfpy201810-123
(remember to “fork” it when you open it)

https://link.stthomas.edu/sfpy201810-123 - **Compute Initials**

3A

1. At the end of the program, add:

```
ser1first = df1['First'].str[0]
ser1last = df1['Last'].str[0]
ser1initials = ser1first + '. ' + ser1last + '.'
print(ser1initials)
```

2. Run the code

```
0     J. B.
1     S. C.
2     M. M.
3     C. C.
4     V. S.
5     A. S.
6     A. H.
dtype: object
```

- Do you see the output below?
- Problems? (Remember: chat check)
- What “data types” do you think are in the “ser1…” variables? Rationale? Proof?
- Psychoanalyze my variable names!
  - What might you prefer?
- Could the code take fewer lines?
  - If so, how, and why did I make it so long?
- If we added `print(df1)`, would it show a column with initials?
  - (Feel free to try it after you guess)
- Questions? (Remember: chat check)

3B

https://link.stthomas.edu/sfpy201810-123 - Add "Sorted Series" Initials

1. Backspace out the final `print(ser1initials)`
2. At the end of the program, add:

```
ser1initsrt = ser1initials.sort_values()
print(ser1initsrt)
df1['Initials'] = ser1initsrt
```

3. Run the code

```
6     A. H.
5     A. S.
3     C. C.
0     J. B.
2     M. M.
1     S. C.
4     V. S.
dtype: object
```

- Do you see the output below?
- Problems? (Remember: chat check)
- Note that we added a new "Initials" column to the DataFrame in our variable `df1`, but that we set it to the values of a "sorted" series of initials! (Rows 6, 5, 3, 0, 2, 1, 4!)

  If we added `print(df1)`:
  - What order would the rows of `df1` show up in? 0, 1, 2… or 6, 5, 3…?
  - Would the right initials be attached to the right person?
    - (Feel free to try it after you guess)
- Questions? (Remember: chat check)

Yay! They’re in the right order!

- It must be that “Pandas index magic”

```
      Id    First      Last            Email                      Company Initials
0   5829    Jimmy    Buffet  jb@example.com                          RCA     J. B.
1   2894  Shirley  Chisholm  sc@example.com      United States Congress     S. C.
2    294  Marilyn    Monroe  mm@example.com                          Fox     M. M.
3  30829    Cesar    Chavez  cc@example.com         United Farm Workers     C. C.
4    827  Vandana     Shiva  vs@example.com                     Navdanya     V. S.
5   9284   Andrea     Smith  as@example.com     University of California     A. S.
6    724   Albert    Howard  ah@example.com  Imperial College of Science     A. H.
```

3C

https://link.stthomas.edu/sfpy201810-123 - Add “Sorted List” Initials

1. Backspace out any `print(ser1initsrt)` or `print(df1)`
2. At the end of the program, add:

```
list1initsrt = list(ser1initsrt)
df1['Initials'] = list1initsrt
print(list1initsrt)
```

3. Run the code

```
['A. H.', 'A. S.', 'C. C.', 'J. B.',
'M. M.', 'S. C.', 'V. S.']
```

- Do you see the output below?
- Problems? (Remember: chat check)
- Note that we overwrote the “Initials” column of the DataFrame in our variable `df1`, with “simple list” copy of what was in our “alpha-sorted initials” Series. Lists are always “indexed” a simple “0, 1, 2…”

  If we were to add the code `print(df1)`:
  - Would the right initials be attached to the right person?
    - (Feel free to try it after you guess)
- Questions? (Remember: chat check)

Oh no! We botched the order!

```
      Id    First      Last              Email                     Company Initials
0   5829    Jimmy    Buffet  jb@example.com                          RCA     A. H.
1   2894  Shirley  Chisholm  sc@example.com      United States Congress     A. S.
2    294  Marilyn    Monroe  mm@example.com                          Fox     C. C.
3  30829    Cesar    Chavez  cc@example.com         United Farm Workers     J. B.
4    827  Vandana     Shiva  vs@example.com                    Navdanya     M. M.
5   9284   Andrea     Smith  as@example.com     University of California     S. C.
6    724   Albert    Howard  ah@example.com  Imperial College of Science     V. S.
```

- It's still "Pandas index magic," but our "list" looks like this:

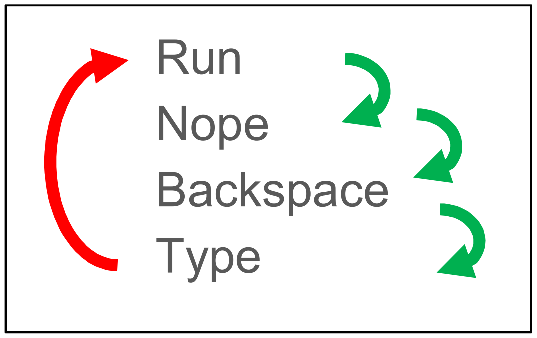
  ```
  ['A. H.', 'A. S.', 'C. C.', 'J. B.', 'M. M.', 'S. C.', 'V. S.']
  ```

- In the list, "A.H." is "#0" instead of "#6".

  So Pandas put it in "row #0" instead of "row #6."
  - Takeaway: Series & Lists are both 1-D and can be used *somewhat* interchangeably, but not they're not *literally* the same thing.

Lucky Us

- We never overwrote our actual CSV file. 😂



- Pro Tip: Never `.to_csv(…)` to the same file you `.read_csv()` from

More Hands-On

Stay in your current “bunk”

3D

https://link.stthomas.edu/sfpy201810-123 - Sort a whole DataFrame

1. Backspace out all the code we just wrote. Leave all the `.read_csv(…)` and such.
2. At the end of the program, add:

```
df3sorted = df3.sort_values(by=['D.O.B.'], ascending=[True])
print(df3sorted[['First','Last','D.O.B.']])
```
← *(note the DOUBLE square brackets!)*

3. Run the code.

- We were trying to display people from oldest to youngest.
  - Did we do that?
  - If not, what did we actually do, and conceptually, what might help?
- Problems? Questions? (Remember: chat check)

```
      First       Last      D.O.B.
1  Quintina       Lean  10/14/1963
8      Kata     Windus   10/4/1991
3      Yuri     Dalton  11/12/1980
2     Corny     Noller  12/13/1990
0     Salli     Broxup   12/3/1991
5      Mata Pierrepont   8/19/1970
6   Othelia   Eastbury    8/4/1955
7     Pansy     Mallya    8/4/1955
4   Doretta     Herche   9/21/2010
```

Interpreting “D.O.B.” as a date: 2 approaches

Tell `.read_csv(…)` to interpret it as a date

Pros:

- Short & sweet if just exploring
- Auto-fixes “D.O.B.” to YYYY-MM-DD for `.to_csv(…)` if that’s okay

Cons:

- Have to fix “D.O.B.” back before `.to_csv(…)` if we *liked* m/d/yyyy

Add “DOBdate” column, sort on “DOBdate,” don’t `.to_csv(…)` “DOBdate”

Pros:

- More granular control
  - e.g. “timezone” plugins that help convert a “local” timestamp to a “UTC” timestamp without tripping over Daylight Svgs. Time

Cons:

- More lines of code

We don’t always need a sortable date

- Not doing anything to D.O.B.
- Grouping by D.O.B. (as long as it’s distinct, it’ll do)

3E

https://link.stthomas.edu/sfpy201810-123 - Sort a DataFrame by date

1. Don't erase any code from the last exercise – we're going to fix code above it so that it'll work right.
2. Near the top of our code, find the `df3 = pandas.read_csv(…)` line and, right after the word "object," add `, parse_dates=['D.O.B.']` *(with the leading comma)* so that the line ends up looking like this:

```
df3 = pandas.read_csv('https://(…long URL here…).csv', dtype=object, parse_dates=['D.O.B.'])
```

3. Run the code.

- Are you seeing people sorted old->young?
- We did the "1st approach".
  - Note that the DOB looks different (now YYYY-MM-DD)
- Problems? Questions? (Remember: chat check)

```
      First        Last     D.O.B.
6   Othelia    Eastbury 1955-08-04
7     Pansy      Mallya 1955-08-04
1  Quintina        Lean 1963-10-14
5      Mata  Pierrepont 1970-08-19
3      Yuri      Dalton 1980-11-12
2     Corny      Noller 1990-12-13
8      Kata      Windus 1991-10-04
0     Salli      Broxup 1991-12-03
4   Doretta      Herche 2010-09-21
```

Treat-filled Q&A!

(Instead of a break yet – sorry.)

Useful yet?

Imagine you've already mastered what you've seen.

Sorting rows, adding/discarding columns, discarding/keeping duplicate rows, counting unique/duplicate rows & values…

Any "Python beats Excel" use cases yet?

You won't hurt my feelings!

(Remind me to repeat for online & read chat)

Lecture: “Column” Actions

(No need to memorize – we’ll practice)
[[102 breakpoint-ish (2:20)]]

Selecting specific columns of a DataFrame

- You've already seen this "bracket notation."
  - `yourDataFrameHere['SomeColumnName']` is an **expression** that produces a **new** "**Series**" representing that column. e.g.

    `df1['First']`

  - `yourDataFrameHere[['Col1','Col2','Col3']]` is an **expression** that produces a **new** "**DataFrame**" representing just those columns. e.g.

    `df3sorted[['First','Last','D.O.B.']]`

    - Pro Tip: Useful for "peeking" at "wide" tables, like `.head()` is with "long" tables.
      - (Yes, `df3sorted[['First','Last','D.O.B.']].head()` works!)
    - Note: `['Col1','Col2','Col3']` is just a standard Python "**list**" expression.

Modifying specific columns of a DataFrame

- "DataFrame-Bracket" notation has **special behavior** on the left side of an `=`
  - `df1['First'] = 'Anush'` will **modify the contents of the DataFrame** stored in the variable "df1," **overwriting everyone's first name to "Anush."**
    - Or, if there is no column named "First," this statement adds a "First" column and populating it all the way down with "Anush."
      - Very handy for, say, adding "CampaignId" to a CSV file.
      - We leveraged this earlier with `df1['Initials'] = …`
  - Double-bracket notation is similar, except erroring instead of adding nonexistent columns.
    - `df1[['First','Last']] = ['Anush','Lopez']` will turn everyone into an "Anush Lopez"
    - `df1[['First','Last']] = 'Kelly'` will turn everyone into a "Kelly Kelly"

DataFrame-Bracket Notation Power Use

- We did this in several steps with the “initials” exercise, saving off our intermediate “series” into variables for legibility. Here’s a similar one-liner.

```
df1['Full'] = df1['Last'] + ', ' + df1['First']
```

- `df1['Full']` is serving the special function of modifying “df1”

- `df1['Last']` and `df1['First']` are just **expressions** that produce **brand new** “Series”-typed results *(typing them doesn’t modify “df1”)*

Adding empty columns (e.g. to fill in later)

```
yourDF['NewColumnName'] = None
```

- The “None” keyword, with a capital N, Python’s special “NULL” value.
  - With Pandas, you might also see “NaN”. Same idea.
    Technically different; I haven’t had to care.
    - Both reply “True” to `.isnull()`
    - Both write a blank cell when exporting to CSV

Renaming columns *(e.g. “Id”->“ContactId”)*

```
yourDF.rename(columns={'Old1':'New1','Old2':'New2'})
```

- This **expression** merely produces a ***new copy*** of “yourDF” with the column names “Old1” & “Old2” replaced by “New1” & “New2,” respectively.
    - Can be handy with `.merge(…)` (VLOOKUP) operations
- To ***actually change*** the contents of “yourDF,” do either of these **statements**:
    - `yourDF.rename(columns={'Old1':'New1','Old2':'New2'}, inplace=True)`
    - `yourDF = yourDF.rename(columns={'Old1':'New1','Old2':'New2'})`

Selecting “all but” specific columns

```
yourDF.drop(columns=['Unloved1','Unloved2'])
```

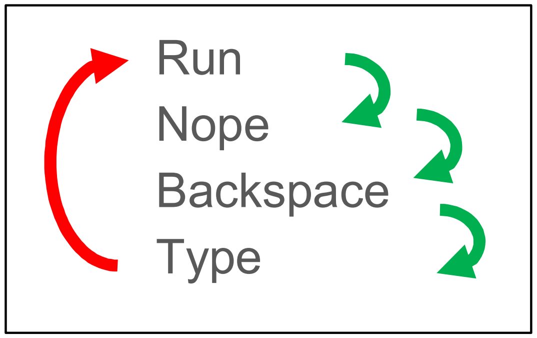
- This **expression** merely produces a ***new copy*** of “yourDF” with all columns *except* “Unloved1” & “Unloved2.”
  - Older versions of Python like in our “Codebunk” environment require one of these instead:
    - `yourDF.drop(['Unloved1','Unloved2'], axis='columns')`
    - `yourDF.drop(['Unloved1','Unloved2'], axis=1)`

- To ***actually change*** the contents of “yourDF,” do either of these **statements**:
  - `yourDF.drop(columns=['Unloved1','Unloved2'], inplace=True)`
  - `yourDF = yourDF.drop(columns=['Unloved1','Unloved2'])`

Expressions vs. Statements review

- Note how an "**expression**" (something that *is* something – it doesn't *do* anything) can suddenly become a "**statement**" (something that *does* something – it *isn't* a value that you can `print(…)`) with a teeeeeny bit of code like "`, inplace=True`".

- If your program is acting weird, keep this in mind and:



Series Transformations

- Every “Series” inherently has a bazillion `.somethingOrOther…` operations that can follow directly after it *(no space)*.
    - Some of them aggregate the cells of the Series (e.g. “max” or “sum” type operations)
    - Most of them iterate over every cell in the Series, doing the same thing to each one.
        - We used one of these earlier to grab “character #0” of df1’s “Last” column:

          ```
          df1['Last'].str[0]
          ```
        - Typically, they produce a **new** Series that’s an altered *copy* of the input Series.
        - Some of them will error out if they hit a cell of a nonsensical “data type” for the operation.

Series Transformations

- Seriously. There are hundreds. https://pandas.pydata.org/pandas-docs/stable/api.html#series
    - Just the several dozen text-manipulating ones: https://pandas.pydata.org/pandas-docs/stable/api.html#string-handling
- Under construction by me: a “frequently useful” shortlist at: https://pypancsv.github.io/pypancsv/commonoperations/

Series Transformations – Boolean Series

- Probably the most useful kind of “series transformations” are the ones that produce a new Series full of True/False (“Boolean”) values.
- The “False” values in such Series let **you “skip over” corresponding rows** of a DataFrame or another Series **while performing some action**.

Hands-On: More Row Filters

Re-visit https://link.stthomas.edu/sfpy201810-123 anew
(remember to “fork” it when you open it)

We’ve done a few filters w/o “True/False Series”

So far, we’ve done:

`.head(…)`

`.tail(…)`

`.drop_duplicates(…)`

3F

https://link.stthomas.edu/sfpy201810-123 - **Identify redundant rows**

1. At the end of the program, add:

```
print(df3.duplicated(keep=False))
```

2. Run the code

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
dtype: bool
```

- Do you see the output below?
- Problems running? (Remember: chat check)
- What "data type" is this? Rationale? Proof?
- keep=False makes this operation return True for a row if it's "like" any other row.
- We didn't specify any columns, so it's looking at all columns ("pure duplicates").
- Questions? (Remember: chat check)

3G

https://link.stthomas.edu/sfpy201810-123 - Identify redundant rows

1. Change your previous line to:

```
print(df3.duplicated(['Address','D.O.B.'], keep=False))
```

2. Run the code

```
0     False
1     False
2     False
3     False
4     False
5     False
6      True
7      True
8     False
dtype: bool
```

- Do you see the output?
- Problems? (Remember: chat check)
- It looks like “row 6” & “row 7” have a “twin roommate” somewhere in the data set. (Probably each other, since they’re the only 2…)
- What if this were 8,000 rows? How would we know if any were “True?”
    - Ideas? (Hint: “Power of One”)
- Questions? (Remember: chat check)

https://link.stthomas.edu/sfpy201810-123 - Count redundant row 3H

1. Change your previous line to:

```
print(df3.duplicated(['Address','D.O.B.'], keep=False).sum())
```

2. Run the code

- Do you see the number **2** as output?
- Problems? (Remember: chat check)
- What was that black magic?
  - Ideas?
- Questions? (Remember: chat check)

Magician's Secrets: Do Duplicates Exist?

```
df3.duplicated(['Address','D.O.B.'], keep=False).sum()
```

- `df3` is a dataframe, which means it has a `.duplicated(…)` operation.
- The output of that operation is a True/False-filled Series.
- All Series have a `.sum()` operation that will add up the value of all of their cells … presuming those cells are numeric.
- It turns out that Pandas is happy to treat True/False as 1 & 0, meaning that the "sum" is a record-count of "True" values in the series.

Yay – now we can quick-check whether duplicates exist in an 8,000-row CSV file.

Now let's see them.

3l

https://link.stthomas.edu/sfpy201810-123 - Display redundant rows

1. Backspace out your code from the last exercise.
2. Add the following code to the end of the program:

```
ser3isdup = df3.duplicated(['Address','D.O.B.'], keep=False)
print(df3[ser3isdup])
```

3. Run the code

```
      Id    First      Last    D.O.B.                          Address
6  32443  Othelia  Eastbury  8/4/1955  87834 Lyons Terrace, Rainy, OR
7  22082    Pansy    Mallya  8/4/1955  87834 Lyons Terrace, Rainy, OR
```

- Do you see the output?
- Problems? (Remember: chat check)
- What "data type" is this? Rationale? Proof?
- Have we seen this `someDataFrame[…]` syntax before?
- Questions? (Remember: chat check)

Lecture:
“Row Filter” Actions

- You just saw a NEW flavor of “bracket notation.”
  - `yourDataFrameHere[someTrueFalseSeriesWithTheSameRowIDs]` is an **expression** that produces that produces a **new** “**DataFrame**” representing just the rows where `someTrueFalseSeriesWithTheSameRowIDs` was “True.” 2 examples:
    - `df3[df3.duplicated(['Address','D.O.B.'], keep=False)]`
    - `df3[ser3isdup]`
  - I prefer #2! Yay, variables.

- Because any `yourDF[someSeries]` expression is itself a DataFrame, that means *it too* has “standard bracket notation” for “**column selection**.”
    - `df3[ser3isdup]['First']` is an **expression** that would give us a **new** 2-item “**Series**”, with row IDs 6 & 7, showing “Othelia” & “Pansy.”
    - `df3[ser3isdup][['First','Last']]` is an **expression** that would give us a **new** 2-column, 2-row “**DataFrame**,” with row IDs 6 & 7, showing “Othelia Eastbury” & “Pansy Mallya.”
    - You can’t `[]` forever like that. At some point, Python will yell at you for being ambiguous.
        - However, you often *can* “checkpoint” what you’ve made by saving it into a variable and then pick up from there as usual.
    - Python will yell at you if you try to put either of these onto the *left*-hand side of an equation. Unfortunately, they’re not for selectively editing cells of a DataFrame.

If Pandas doesn't yet "all look alike" enough…

- `df[someTrueFalseSeriesSameLength][someColNameOrList]` → "**DataFrame**" w/ rows where T/F series=True; cols. as specified. **Not editable** left of "=". Sdlkf …
  - `df3[ser3isdup][['First','Last']]` gives a 2-column, 2-row "**DataFrame**," w/ row IDs 6 & 7, showing "Othelia Eastbury" & "Pansy Mallya."
- There's an unrelated `df[…][…]`. Yay. 😱 😓 😭
  - `df[someSingleColName][someRowIdOrList]` → "**Series**" of specified col., w/ items indicated by row ID. **Editable** left of "=" *(Level 102)*
    - e.g. `df3['Last'][[5,7]]` → 2-item "**Series**," #5: "Pierrepont" & #7: "Mallya"
  - `df[someSingleColName][someTrueFalseSeriesSameLength]` → "**Series**" of specified col., w/ items where T/F series=True. **Editable** left of "=" *(Level 102)*
    - e.g. `df3['Last'][ser3isdup]` → 2-item "**Series**," #6: "Eastbury" & #7: "Mallya"
  - Doesn't work w/ column name list *(will yell at you)*.
  - Not editable if column doesn't exist yet in `df` *(will yell at you)*.

Door Prize: A Script!

(And then a break)

A real-life script “finddupes.txt”

- I like to save my favorite Python scripts for future reference. Here’s one:

```
import pandas
pandas.set_option('expand_frame_repr', False)
filename = 'c:\\example\\sample.csv' # Edit this before running
dupeColumns = ['col1','col2','col3'] # Edit this before running
df = pandas.read_csv(filename, dtype=object)
isDupeSeries = df.duplicated(dupeColumns, keep=False)
isFirstDupeSeries = df.duplicated(dupeColumns, keep='first')
print(str(isDupeSeries.sum()) + ' dupes in ' +
      str(isFirstDupeSeries.sum()) + ' groups in ' +
      str(len(df)) + ' rows')
print('\r\n---The duped rows are:---')
print(df[isDupeSeries])
print('\r\n---The "dupe keys" are:---')
print(df[isFirstDupeSeries][dupeColumns])
```

Questions? (Chat room?)

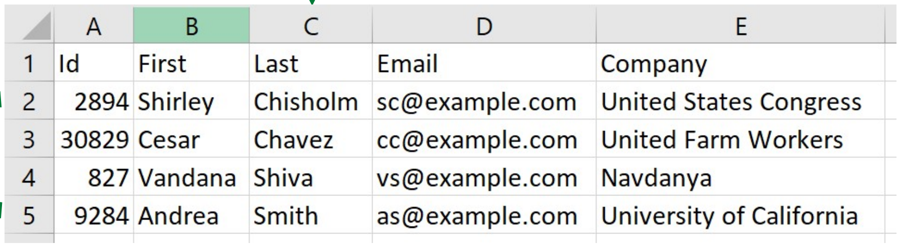
10-Minute Break

Questions? (Chat room?)

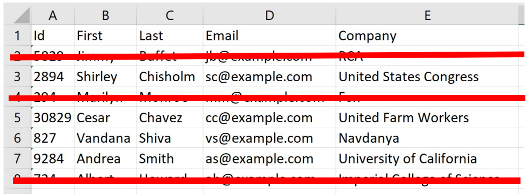
Lecture: “Starts With” Row Filtering

```
➢ 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~~

Hands-On:
Fancier Row Filter

https://link.stthomas.edu/sfpy201810-filter1

🍴 *(remember to "fork" it when you open it)* 🍴

https://link.stthomas.edu/sfpy201810-filter1

4A

- Any problems running it?
    - (Remind me to check the chat)

https://link.stthomas.edu/sfpy201810-filter1

4B

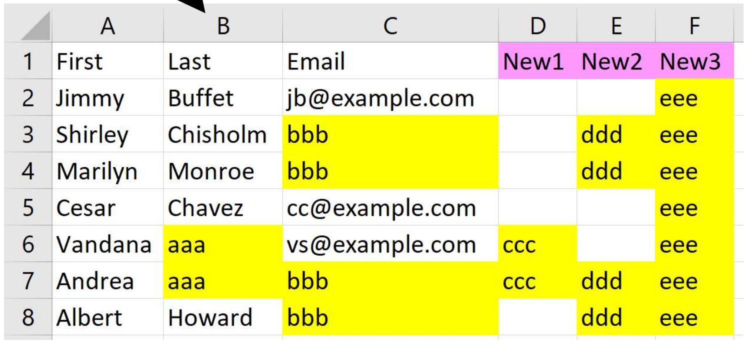
Hands-On: Together *(come up to my computer!)*, we'll edit the code so that

- Instead of doing:
  - 'Display all columns, but only rows where "Last" starts with capital "C" or "S"'
- It will do:
  - 'Display all columns, but only rows where "Company" case-insensitively ends with "a" **or where** "Id" is less than 800'
- Hint: Every Series has the following operations:
  - `.str.lower()` *(the resulting output is also a Series, full of text-typed data)*
  - `.str.upper()` *(the resulting output is also a Series, full of text-typed data)*
  - `.str.endswith(…)` *(the resulting output is also a Series, full of True-False data)*
  - `.astype('int')` *(the resulting output is also a Series, full of integer-typed data)*

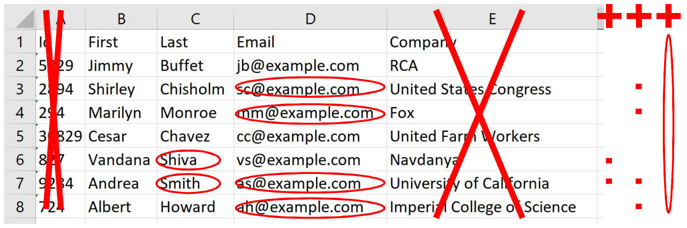
FOR POSTERITY: Copy/paste our code below.

“102” taster: editing cells based on existing data

```
➢ theseRowsLastNamesStartWithCapitalS = df1['Last'].str.startswith('S')
➢ theseRowsHaveA4InTheirId = df1['Id'].astype(str).str.contains('4')
➢ df1['Last'][theseRowsLastNamesStartWithCapitalS] = 'aaa'
➢ df1['Email'][theseRowsHaveA4InTheirId] = 'bbb'
➢ df1['New1'] = None
➢ df1.loc[theseRowsLastNamesStartWithCapitalS, 'New1'] = 'ccc'
➢ df1['New2'] = None
➢ 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

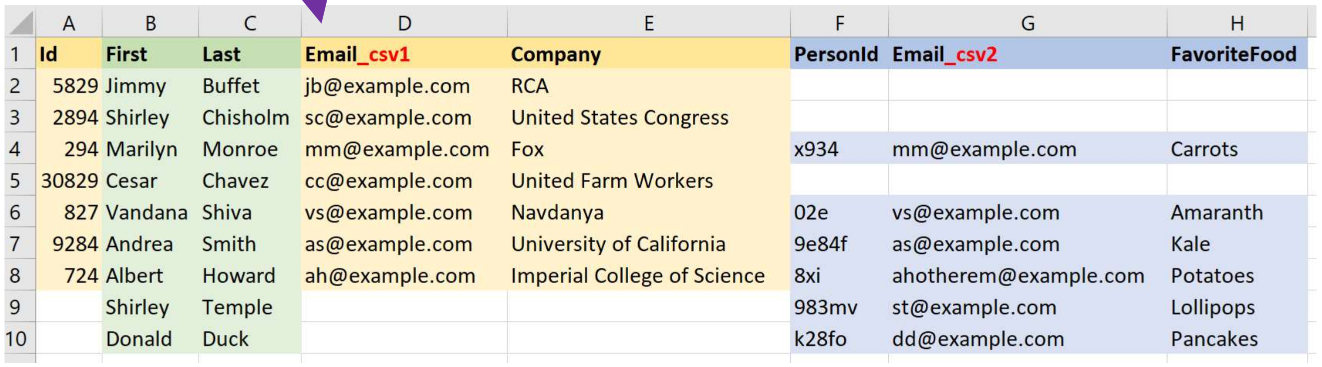


	A	B	C	D	E
1	Id	First	Last	Email	Company
2	5[illegible]29	Jimmy	Buffet	jb@example.com	RCA
3	2[illegible]94	Shirley	Chisholm	sc@example.com	United States Congress
4	2[illegible]4	Marilyn	Monroe	mm@example.com	Fox
5	3[illegible]829	Cesar	Chavez	cc@example.com	United Farm Workers
6	8[illegible]7	Vandana	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

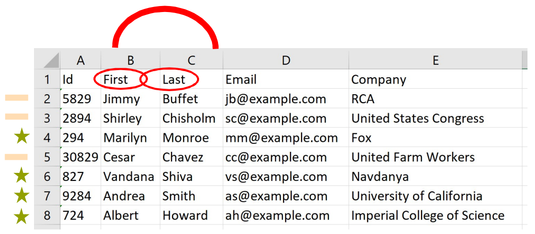
https://link.stthomas.edu/sfpy201810-democomplexcellupdates

“102” taster: 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)
```

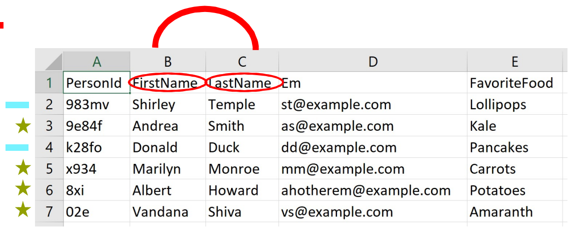


	A	B	C	D	E	F	G	H
1	Id	First	Last	Email_csv1	Company	PersonId	Email_csv2	FavoriteFood
2	5829	Jimmy	Buffet	jb@example.com	RCA			
3	2894	Shirley	Chisholm	sc@example.com	United States Congress			
4	294	Marilyn	Monroe	mm@example.com	Fox	x934	mm@example.com	Carrots
5	30829	Cesar	Chavez	cc@example.com	United 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

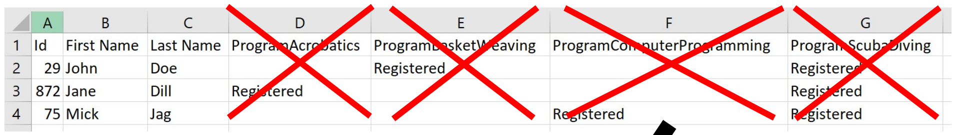
+



		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

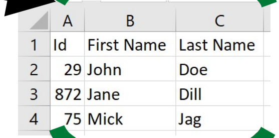
Advanced Demo: “generators,” part 1

- Weird but concise code for:
  “Show ‘mydf,’ just columns that don’t start with the phrase ‘Program.’”



	A	B	C	D	E	F	G
1	Id	First Name	Last Name	~~ProgramAcrobatics~~	~~ProgramBasketWeaving~~	~~ProgramComputerProgramming~~	~~ProgramScubaDiving~~
2	29	John	Doe		~~Registered~~		~~Registered~~
3	872	Jane	Dill	~~Registered~~			~~Registered~~
4	75	Mick	Jag			~~Registered~~	~~Registered~~

```
➢ colsThatStartWithProgram = [x for x in mydf.columns if x.startswith('Program')]
➢ mydf = mydf.drop(colsThatStartWithProgram, axis='columns')
➢ mydf.to_csv('C:\\yay\\out_generator1.csv', index=False, quoting=1)
```

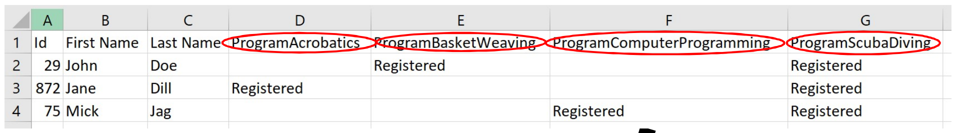


	A	B	C
1	Id	First Name	Last Name
2	29	John	Doe
3	872	Jane	Dill
4	75	Mick	Jag

https://link.stthomas.edu/sfpy201810-demogenerator1

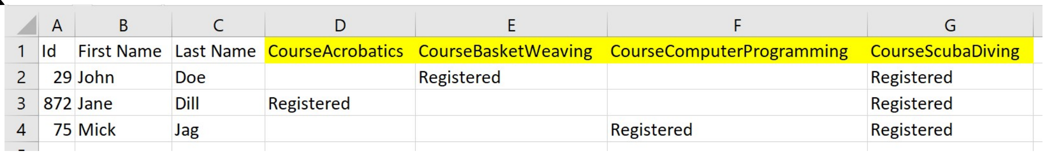
Advanced Demo: “generators,” part 2

- Weird but concise code for:
  “Rename any column that starts with ‘Program’ to ‘Course…’”



	A	B	C	D	E	F	G
1	Id	First Name	Last Name	ProgramAcrobatics	ProgramBasketWeaving	ProgramComputerProgramming	ProgramScubaDiving
2	29	John	Doe		Registered		Registered
3	872	Jane	Dill	Registered			Registered
4	75	Mick	Jag			Registered	Registered

```
➢ colsThatStartWithProgram = [x for x in mydf.columns if x.startswith('Program')]
➢ renameKey = {x:x.replace('Program','Course', 1) for x in colsThatStartWithProgram}
➢ mydf = mydf.rename(columns=renameKey)
➢ mydf.to_csv('C:\\yay\\out_generator2.csv', index=False, quoting=1)
```



	A	B	C	D	E	F	G
1	Id	First Name	Last Name	CourseAcrobatics	CourseBasketWeaving	CourseComputerProgramming	CourseScubaDiving
2	29	John	Doe		Registered		Registered
3	872	Jane	Dill	Registered			Registered
4	75	Mick	Jag			Registered	Registered

https://link.stthomas.edu/sfpy201810-demogenerator2

102: “State code vs. Label typo-hunt”

- Let’s say you have a 2-column table of “Unique IDs” and “Country Names.”
- You want to dummy-check that no country is listed twice.
- Let’s peek at https://link.stthomas.edu/sfpy201810-demostatetypo

```
uniqueColBPerColA = someDF.groupby(['colNameA'])['colNameB'].nunique()
print(uniqueColBPerColA[uniqueColBPerColA>1])
```

Links & Resources

- https://tinyurl.com/pypancsv - All my notes, slides, etc. so far
  - Slides *(once I get them online)*
  - Examples and exercises
  - “Commonly Used Operations”
    - (Under development … I promise it’ll get better!)

- https://tinyurl.com/PyPanCsvWinIde - getting an “IDE” onto your computer

- https://pbpython.com – “Practical Business Python”
  - (as with many blogs, might be best to start by browsing older posts)