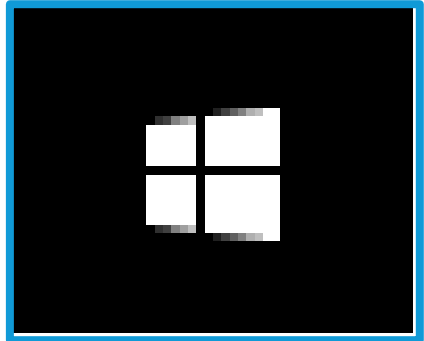


Python for Spreadsheet Manipulation 102



	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!)

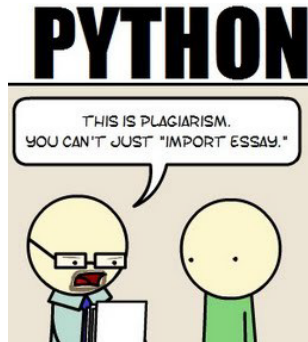
What do YOU wish you
could do?

(Remind me to repeat for the microphone, please)

Syllabus

102 (today)

- 101 recap
- Coding
 - Adding/dropping columns
 - Basic matching / VLOOKUP
 - Row Filtering
- If time:
 - Selectively editing cells based on column+row filter combinations
 - A simple pivot?



101 Recap

Coding 101

Pieces of Code: Expressions vs. Statements

Expressions

- 1
- 1+3
- 'Hi There'
- NULL
- True
- False
- $2 < 1$
- `type('Hi There')`

- **Like Excel formulas**
- Nestable! (Like in Excel ... learn to pick apart sub-parentheses for understanding!)
 - $(1 + 3) < (2 / 3)$

Statements

- `print(1+3)`
- `coolVariableName = 1+3`
- `print(4)`

- **Statement:Program :: Sentence:Essay**
- 1 per line

Pieces of Code: Variables

- Nicknames you can “assign” values to for later reference in your code (by typing to the left of “=”)
 - e.g. **codeThatStandsInForTheNumberTwo = 1 + 1**
- Pick just about anything, except:
 - Something that’s already a Python command
- Do NOT surround the variable name with quotes ...
 - You surround words with quotes when you’re trying to say that they’re text, not code.
 - Variable names BECOME valid code.



Working Blind

- When coding, you don't get to SEE the results of your computations unless you save it to a file on your hard drive or do "print()" in a context that actually shows you output from "print()" (like an "IDE")
 - FAST!
 - But new paradigm



Hands-On: Running & Writing Code

- Step 1: Open <https://link.stthomas.edu/sfpy201901-hello>
- Step 2: Hit the big green “run” button, top center
- Step 3: Do you see the text “Hello, world!” at right?

- Step 4: At left, change “**Hello, world!**” to say “**Yay, us!**”
(Leave the single quotes in place. They distinguish code from text.)
- Step 5: Hit the big green “run” button, top center.
- Step 6: Do you see the text “Yay, us!” at right?


DataFrames (tables), Series (lists...-ish), & 0-D data

- Python's "Pandas" plugin's main commands process entire abstractions of a spreadsheet, or of one of its rows/columns, at a time, unlike Excel, which is more "cell-by-cell" oriented.
- Be able to think through whether a given piece of data you're trying to process would be, to the computer, a 2-D table, a 1-D row/column/list, or a 0-D single point (e.g. the kind of things that belongs in a single cell).
If a single point, what's its "data type?" (Just like in Salesforce, integers & text & dates & such are all computed differently.)
 - Thinking about entire tables and rows as a single data point is DIFFERENT from Salesforce but normal in programming. Closest Excel analogy: a range.

Anatomy of a DataFrame / Series

- DataFrames:


- Rows indexed by “Row IDs” (typically starts w/ “0” & counts up from what was originally first line before any sorting).
- Columns indexed by column names



	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

- Series:

- Items indexed by “Item IDs”
 - If the Series represents a column, this will probably be the “row IDs” of the rows the cell values came from
 - If represents a row, the “column IDs” of the columns the cell values came from



```
0    Shirley
1    Andrea
2    Donald
3    Marilyn
4    Albert
5    Vandana
Name: FirstName, dtype: object
```



```
PersonId      k28fo
FirstName     Donald
LastName      Duck
Em            dd@example.com
FavoriteFood   Pancakes
Name: 2, dtype: object
```

The “type()” operation is THE BEST



<pre>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!')</pre>	<pre>Hello World <class 'str'> 5 <class 'int'> None <class 'NoneType'> False <class 'bool'> 30.0 <class 'float'> False <class 'bool'> {{{nothing prints out for this line}}}} 30.0 <class 'float'> False <class 'bool'> Bye!</pre>
--	--

Hands-On: Reading & Manipulating CSV Files

2

- Step 1: Open <https://link.stthomas.edu/sfpy201901-files>
- Step 2: Remind me to explain the code that was already on screen.
- Step 3: Add a new line to the end of the file and type these **4 lines** exactly as seen here (*hitting “enter” to start a new line as indicated*):

```
df1 = pandas.read_csv(filepath1)
print(df1)
print('-----')
print('There are ' + str(len(df1)) + ' rows')
```

- Step 4: Hit the big green “run” button, top center
- Step 5: Do you see a table full of contacts at right, then a divider line, then an announcement that there are “7 rows”?

“Pandas” Commands We Practiced

Store a copy of a CSV file's contents in a variable (whose "data type" will be a "Pandas DataFrame")

- `import pandas`
- `dfVarName1 = pandas.read_csv('c:\\yay\\inputfile.csv')`
- `dfVarName2 = pandas.read_excel('c:\\yay\\inputfile.xlsx')`

Grab a single column from a table

- If the table (data type: “DataFrame”) is stored in a variable called “**df4**” and looks roughly like this:

	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

- Then the “expression” to refer to just the contents of its “Program Registered For” column is (output data type: “Series”)

```
df4['Program Registered For']
```

	A
1	Program Registered For
2	BasketWeaving
3	ScubaDiving
4	ScubaDiving
5	Acrobatics
6	ScubaDiving
7	ComputerProgramming

- But remember, it vaporizes as soon as it’s computed unless we save it into a new variable, print() it, write it out to a file on our hard drive, etc.

Fun tricks w/ `df['colName']`

<https://link.stthomas.edu/sfpy201901-101recap>

1. `print(list(df['colName'].unique()))`
2. `df['colName'].to_csv('c:\\yay\\just_this_col.csv', index=False, header=True)`
3. `print(df['colName'].unique())`
4. `print(len(df['colName'].unique()))` ← `len(...)` vs. `...unique()`—quirk; read doc!
5. `print(sorted(df['colName'].unique(), key=str.lower))`
6. `df['colName'] = df['colName'].str.upper()`
7. `df['newCol1'] = df['colName'].str.upper()`
8. `df['newCol1'] = 'Kelly'`
9. `df['newCol1'] = None`

Grab a sub-table from a table

- If the table (data type: “DataFrame”) is stored in a variable called “**df4**” and looks roughly like this:

	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

- Then the “expression” to refer to just the contents of its “First Name” & “Last Name” columns is (output data type: “DataFrame”)

```
df4[['First Name', 'Last Name']]
```

	A	B
1	First Name	Last Name
2	John	Doe
3	John	Doe
4	Jane	Dill
5	Jane	Dill
6	Jane	Dill
7	Mick	Jag

- Note the extra square brackets! The inner ones indicate a Python LIST.

Fun tricks w/ `df[columnList]` ← 2nd brackets?!

<https://link.stthomas.edu/sfpy201901-101recap>

1. `df[columnList].to_csv('c:\\yay\\just_this_col.csv', index=False)`
2. `namecols = [x for x in df4.columns if 'Name' in x]`
`print(namecols)`
`print(df4[namecols])`
3. `df[['First', 'Last']] = 'Kelly'` ← fill down as “Kelly Kelly”
4. `df[['First', 'Last']] = ['John', 'Smith']` ← fill down
5. `df[['First', 'Last']] = df[['Last', 'First']]` ← swap VALUES
6. `df = df[['Last', 'First']]` ← subselect / reorder COLUMNS
7. Note no “new column names” “fill down”!

Yikes! That's a lot of square-bracket subtlety!

- Q: How to deal?

A: DON'T OVERWRITE YOUR INPUT CSV FILENAME
ON YOUR HARD DRIVE.

- Don't “.to_csv(...)” to the same “...” that you did “pandas.read_csv(...)” from.

Once you've assured that...

- Play a lot with `print(...)`, `print(type(...))`, `print(...head())`, etc!
 - (`print(dfVariableName.head())` grabs the 1st 5 rows of DataFrame-typed data)

- Q: Why the madness?
- A: To make ultra-common commands concise to type once you know them.

Feeling overwhelmed, non-101-ers?

- Don't!
- Just follow instructions today and go to <https://pypancsv.github.io/pypancsv/HandsOn201810/> afterwards to experiment hands-on with everything I just showed at your leisure, for better retention.

- Just a few more...

Just a few more... (not shown in runnable code)

1. `df.head(2)`

Produces a copy of your DataFrame, containing only its first 2 rows

2. `df.tail(2)`

Produces a copy of your DataFrame, containing only its last 2 rows

3. `someSeries.head(2)`

Produces a copy of your Series, containing only its first 2 items

4. `someSeries.tail(2)`

Produces a copy of your Series, containing only its last 2 items

5. `someList[4] ← new!`

Produces single data-point, containing only your List's 5th item (count starts @ 0)

6. `someList[:2] ← new!`

Produces a copy of your List, containing only its first 2 items

7. `someList[-2:] ← new!`

Produces a copy of your List, containing only its last 2 items

8. All the “`someList`” tricks work with “`somePieceOfText`” data, only substitute “letter”/“character” for “item”. You'll see `someText[0]` a lot for grabbing initials!

Last ones...

<https://link.stthomas.edu/sfpy201901-101recap>

1. `df.drop_duplicates(subset=columnList, keep='first')`
Produces a copy of your DataFrame, minus all “redundant” rows
2. `df.drop_duplicates(subset=columnList, keep=False)`
Produces a copy of your DataFrame, KEEPING ONLY “special snowflake” rows
3. `hasADupeTFSeries = df.duplicated(subset=columnList, keep=False)`
(Right-hand side produces a True/False “Series”; full code adds it as a new column to your original DataFrame.) (Yes, the “keep=False” wording here is counter-intuitive.)
4. `firstOfADupeSetTFSeries = df.duplicated(subset=columnList, keep='first')`
(Same idea as #4, but flags whether row is “first of a dupe-set”)
5. `df[hasADupeTFSeries] ← df[...]` where ... is a same-IDs “Series” of true/false shows only “true” rows
(Produces a copy of your DataFrame, MINUS “special snowflake” rows, if did #3.)
6. `'There are ' + str(hasADupeTFSeries.sum()) + ' duplicated records in ' + str(firstOfADupeSetTFSeries.sum()) + ' groups'`
(Yay, “Power of One!” True = 1, False = 0. Presumes you did #3 & #4.)

Questions? (Chat room?)

10-Minute Break

Columns

Adding, Deleting, Reordering, & Renaming

(useful before/after combining spreadsheets!)

Add

1. `dfVarName['ColumnName'] = ...`

👉 Standard

👉 Spaces: `'ColumnName' / 'Column Name'`

🚫 Only add 1 column at a time

🚫 Permanent change to the table stored in `dfVarName`

2. `dfVarName = dfVarName.assign(ColumnName1 = ..., ColumnName2 = ...)`

👉 Add multiple columns

👉 `dfVarName.assign(...)` is just a copy of `dfVarName`.

📄 Instead of overwriting the contents of `dfVarName`, you can use that expression inside another expression, and when you later referred to `dfVarName`, it would be unchanged.

(Handy for “which spreadsheet?” labels in “concat” operations.)

🚫 No spaces: `'ColumnName'`

Delete

1. `dfVarName = dfVarName[ListOfColumnNamesHere]`

📘 Delete all columns *except* those in `ListOfColumnNamesHere`

👉 Handy if you're only keeping a few columns.

👉 🚫 Columns now arranged in the order you listed them in `ListOfColumnNamesHere`

2. `dfVarName = dfVarName.drop(columns=ListOfColumnNamesHere)`

📘 Deletes the columns in `ListOfColumnNamesHere`

👉 Handy if you're only deleting a few columns

👉 🚫 Does not change the order of the remaining columns

Both `dfVarName[...]` and `dfVarName.drop(...)` are just copies of `dfVarName`.

📘 Instead of overwriting the contents of `dfVarName`, you can use that expression inside another expression or save it to a different variable, and when you later referred to `dfVarName`, it would be unchanged.

**(Remember to use `[...,...,...]` to indicate "this is a list!" when typing `ListOfColumnNamesHere` if not using a variable holding a list.)*

Reorder

1. `dfVarName = dfVarName[ListOfColumnNamesHere]`

- ⓘ Ensure `ListOfColumnNamesHere` includes every column name in `dfVarName`. Otherwise, you'll **also** delete columns not named!

`dfVarName[...]` is just a copy of `dfVarName`.

- ⓘ Instead of overwriting the contents of `dfVarName`, you can use that expression inside another expression or save it to a different variable, and when you later referred to `dfVarName`, it would be unchanged.

Note: Reordering is for human eyes. Computer doesn't really care.

**(Remember to use `[...,...,...]` to indicate "this is a list!" when typing `ListOfColumnNamesHere` if not using a variable holding a list.)*

Rename

```
1. df = df.rename(columns={'ColumnName1':'NewColumnName1','ColumnName2':'NewColumnName2'})
```

👉 Rename multiple columns

dfVarName.rename(...) is just a copy of **dfVarName**.

📄 Instead of overwriting the contents of **dfVarName**, you can use that expression inside another expression or save it to a different variable, and when you later referred to **dfVarName**, it would be unchanged.

Hands-On - <https://link.stthomas.edu/sfpy201901-123>

Go to <https://link.stthomas.edu/sfpy123>

Modify `df1` so that when you put a command `print(df1)` at the end of your code, you get this output:

```
      Hello Last Name                Company First Name      Id
0  Yay Us   Buffet                    RCA      Jimmy   5829
1  Yay Us  Chisholm      United States Congress  Shirley  2894
2  Yay Us   Monroe                    Fox      Marilyn   294
3  Yay Us   Chavez      United Farm Workers     Cesar  30829
4  Yay Us   Shiva                Navdanya     Vandana   827
5  Yay Us   Smith      University of California  Andrea  9284
6  Yay Us   Howard  Imperial College of Science  Albert   724
```

1. Add a column called “Hello” with the phrase “Yay Us” filled in all the way down
2. Rename “Last” to “Last Name” and “First” to “First Name”
3. Delete the “Email” column
4. Reorder the columns to be “Hello,” “Last Name,” “Company,” “First Name,” & then “Id.”

 (<https://link.stthomas.edu/sfpy201901-info#colcommands>)


Hands-On: One Possible Answer

```
df1['Hello'] = 'Yay Us'
df1 = df1.rename(columns={'Last':'Last Name','First':'First Name'})
df1 = df1.drop(columns=['Email'])
df1 = df1[['Hello', 'Last Name', 'Company', 'First Name', 'Id']]
print(df1)
```


1. Did you get the right output?
2. Questions? (Chat room?)

Questions? (Chat room?)

Door Prize – <https://link.stthomas.edu/sfpy201901-info#doorprize-col>



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



	A	B	C	D	E	F	G
1	Acrobatics	BasketWeaving	Computerming	ScubaDiving	Id	First Name	Last Name
2		Registered		Registered	29	John	Doe
3			Registered		75	Mick	Jag
4	Registered			Registered	872	Jane	Dill

```
columnsWithProgramInTheName = [x for x in df.columns if 'Program' in x]
theRestOfTheColumns = [x for x in df.columns if x not in columnsWithProgramInTheName]

newColumnOrder = columnsWithProgramInTheName + theRestOfTheColumns
df = df[newColumnOrder]

renamingMap = {x:x.replace('Program','') for x in columnsWithProgramInTheName}
df = df.rename(columns=renamingMap)
```

Questions? (Chat room?)

Combining Spreadsheets

As Promised!

↕ or ↔ ? It depends on your business problem!

Vertical ↕

- Excel: Copying a spreadsheet/column and pasting it below another one
- Use: Combining equivalent datasets
 - (like 2+ lists of people, or 2+ lists of transaction logs for the same kinds of transactions)

Code:

```
pandas.concat(listOfDataframesOrSeries)
```

Horizontal ↔

- Excel: VLOOKUP
Access: relationship
- Uses:
 - Looking up “reference” data (e.g. the ID for a name) – Excel VLOOKUP
 - Combining info about “the same” people/transactions/etc. from disparate data sources – Access Relationship
- Note: ↔ = “2 at a time”; no 3+

Code:

```
dfVarName1.merge(dfVarName2)
```

↕ (Table or Column) Concatenation ↕

Concatenation ↕ business problems

1. List all unique e-mail addresses in a spreadsheet, whether they be under “Email,” “WorkEmail__c,” or “SchoolEmail__c.”
2. List all unique e-mail addresses between 2 spreadsheets, whether they be under #1’s “Email,” “WorkEmail__c,” or “SchoolEmail__c” fields, or under #2’s “EMAILADDR,” “EMAIL2__C,” “EMAIL3__C,” or “EMAIL4__C” fields.
3. Spreadsheet 1 has columns “First,” “Last,” “Email.”
Spreadsheet 2 has columns “LastName,” “Em,” & “FirstName.”
Concatenate appropriately (e.g. Em = Email) & dedupe (*by all 3 fields together*).
4. Spreadsheet 1, Spreadsheet 2, & Spreadsheet 3 all have columns “Name,” “DOB,” & “AttendedOrNot” (they’re EventBrite exports).
Add a “WhichSheet” column to each of them saying “Event1,” “Event2,” or “Event3,” concatenate, and sort by “Name,” “DOB,” & “WhichSheet.”

↔ (Table) Merge ↔

Merge ↔ business problems

1. Add “Country Code” & “Country Capital” columns to a spreadsheet full of people, using their “MailingCountry” as a matching key to some “Country Detail” spreadsheet’s “Name” column.
2. Combine 2 spreadsheets full of people and things you know about them on “FirstName,” “LastName,” & “Email” as a matching key.
3. Cross-check 2 financial transaction logs that should be identical, ensuring no “transaction ID” exists in only one spreadsheet, nor has a different timestamp between the two spreadsheets.

Hands-On: 3 event rosters, 1 SF-Contact, 1 SF-Campaign

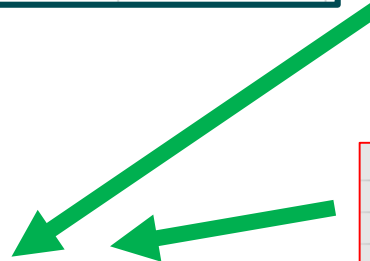


	A	B	C	D	E	F
1	First	Last	Email	Event Name	Event Date	Attendance Status
2	Revkah	Lilburn	rl@example.com	Python for Salesforce 101	10/20/2018	Attended
3	Haskel	Southerns	hs@example.com	Python for Salesforce 101	10/20/2018	No-Show
4	Ermanno	Withinshaw	ew@example.com	Python for Salesforce 101	10/20/2018	Attended

	A	B	C	D	E	F
1	First	Last	Email	Event Name	Event Date	Attendance Status
2	Haskel	Southerns	hs@example.com	Python for Salesforce 101-Office Hours	11/10/2018	No-Show

	A	B	C	D	E	F
1	First	Last	Email	Event Name	Event Date	Attendance Status
2	Julianna	Judron	jj@example.com	Python for Salesforce 102	1/26/2019	No-Show
3	Haskel	Southerns	hs@example.com	Python for Salesforce 102	1/26/2019	Attended
4	Adah	Dimmock	ad@example.com	Python for Salesforce 102	1/26/2019	Cancelled

	A	B	C	D	E
1	ID	FIRSTNAME	LASTNAME	EMAIL	PHONE
2	003X01	Anna	Appleton	aa@example.com	555-555-0101
3	003X02	Adah	Dimmock	dima@example.com	555-555-0202
4	003X03	Ben	Bensalem	bs@example.com	555-555-0303
5	003X04	Julianna	Judron	jj@example.com	555-555-0404
6	003X05	Julianna	Judron	jj@example.com	555-555-0505
7	003X06	Zainab	Zahar	zz@example.com	555-555-0606



	A	B	C
1	ID	NAME	HAPPENED_ON_C
2	701X01	Parasailing Fun Day	7/20/2017
3	701X02	Python for Salesforce 101	10/20/2018
4	701X03	Python for Salesforce 101-Office Hours	11/10/2018
5	701X04	Hockey Outing	1/1/2019
6	701X05	Python for Salesforce 102	1/26/2019

	A	B	C	D	E	F	G	H
1	ContactId	CampaignId	CampaignMemberStatus	Last	First	Email	Event Name	Event Date
2	003X04	701X05	No-Show	Judron	Julianna	jj@example.com	Python for Salesforce 102	1/26/2019
3	003X05	701X05	No-Show	Judron	Julianna	jj@example.com	Python for Salesforce 102	1/26/2019

Hands-On (collaboration out loud encouraged)



4

Go to <https://link.stthomas.edu/sfpy201901-eventmerge> & <https://link.stthomas.edu/sfpy201901-info#ex4> & its cheats.

1. Concatenate the 3 EventBrite sheets vertically ↑ and save it as “eventsdf”
2. Do an “inner” merge from “eventsdf” to “contactsdf” (“inner” implication: drops any attendees not yet in Salesforce – we’ll get to them later) matching on the FIRSTNAME, LASTNAME, & EMAIL; save the result as “merge1df”.
3. Delete columns from “merge1df” so that only the columns of “eventsdf” and the “ID” column remain; ensure the change persists to “merge1df”.
4. Rename the “ID” column of “merge1df” to “ContactId”; ensure “merge1df” changes.
5. Merge “merge1df” against “campaignsdf” on event name & start date; “inner” merge; save the result as “merge2df”.
6. Rename the “ID” column of “merge2df” to “CampaignId”; ensure “merge2df” changes.
7. Rename the “Attendance Status” column of “merge2df” to “CampaignMemberStatus”; ensure “merge2df” changes.
8. Re-order the fields of “merge2df” to be: ContactId, CampaignId, CampaignMemberStatus, Last, First, Email, Event Name, Event Date. Don’t bother including “NAME” or “HAPPENED_ON__C” in your final output if they exist.
9. Export your data to “CampaignMemberRecordsToInsert.csv” and have a look. Does it look like it should?

Hands-On: One Possible Answer

1. Did you get the right output?
2. Questions? (Chat room?)

```
eventsdf = pandas.concat([evdf1, evdf2, evdf3])
mergelfdf = eventsdf.merge(contactsdf, how='inner',
                           left_on=['First','Last','Email'], right_on=['FIRSTNAME','LASTNAME','EMAIL'])
mergelfdf = mergelfdf[list(eventsdf.columns) + ['ID']]
mergelfdf = mergelfdf.rename(columns={'ID':'ContactId'})
merge2df = mergelfdf.merge(campaignsdf, how='inner',
                           left_on=['Event Name','Event Date'], right_on=['NAME','HAPPENED_ON__C'])
merge2df = merge2df.rename(
    columns={'ID':'CampaignId','Attendance Status':'CampaignMemberStatus'})
merge2df = merge2df[['ContactId', 'CampaignId', 'CampaignMemberStatus', 'Last', 'First',
                     'Email', 'Event Name', 'Event Date']]
merge2df.to_csv('CampaignMemberRecordsToInsert.csv', index=False, quoting=1)
```

Door Prize – <https://link.stthomas.edu/sfpy201901-info#doorprize-concat>

“Door Prize Script: Event-Attendance-Concatenating Loop”

- I’ve put code up that can deal with more than 3 “EventBrite” files -- let’s watch it run.

```
import os
import pandas

lookForCSVsInThisFolder = 'c:\\FolderWhereIPutAllTheFiles\\'

listOfDataFrames = []
for x in os.listdir(lookForCSVsInThisFolder):
    if x.endswith('.csv'):
        xdf = pandas.read_csv(x)
        xdf = xdf.assign(WhichCSV = x)
        listOfDataFrames.append(xdf)

concatdf = pandas.concat(listOfDataFrames)

concatdf = concatdf.sort_values(by=['First','Last','Email','WhichCSV'])

concatdf.to_csv('c:\\example\\loopconcat.csv', index=False)
```

Questions? (Chat room?)

10-Minute Break

LEAVE YOUR WORK UP!

Rows

Deleting

Row-Deletion Examples

1. `df[df.duplicated(subset=columnList, keep=False)]`

Produces a copy of your DataFrame, containing only rows that're part of a duplicate set

2. `hasDuplicateSeries = df.duplicated(subset=columnList, keep=False)`

`df[hasDuplicateSeries]` ← *Same as #3, only broken into more lines of code for readability*

3. `df[df['someColumn'].str.upper().str.startswith('S')`

|

`df['Last'].str.upper().str.startswith('C')]`

Produces a copy of your DataFrame, w/ only rows where SomeColumn starts w/ s/S/c/C

4. `lastStartsSSeries = df['SomeColumn'].str.upper().str.startswith('S')`

`lastStartsCSeries = df['SomeColumn'].str.upper().str.startswith('C')`

`lastStartsEitherSeries = lastStartsSSeries | lastStartsCSeries`

`df[lastStartsEitherSeries]` ← *Same as #3, only broken into more lines of code for readability*

Hands-On: Adding Row Filters

When we merged “eventsdf” with “contactsdf,” we let Python drop any “eventsdf” records that *didn’t* have a corresponding “contactsdf.”

Let’s go back and grab *those*, save them as “merge3df,” “insert them into Salesforce” (*we’ll pass them to a fake “Data Loader”*), and carry on otherwise like we did with “merge1df” (eventually merging with campaigns for a “merge4df”).

We’ll export “merge4df” to “CampaignMemberRecordsToInsert2.csv”

	A	B	C	D	E	F	G	H
1	ContactId	CampaignId	CampaignMemberStatus	Last	First	Email	Event Name	Event Date
2	003X61	701X02	Attended	Lilburn	Revkah	rl@example.com	Python for Salesforce 101	10/20/2018
3	003X62	701X02	No-Show	Southerns	Haskel	hs@example.com	Python for Salesforce 101	10/20/2018
4	003X63	701X02	Attended	Withinshaw	Ermanno	ew@example.com	Python for Salesforce 101	10/20/2018
5	003X64	701X03	No-Show	Southerns	Haskel	hs@example.com	Python for Salesforce 101-Office Hours	11/10/2018
6	003X65	701X05	Attended	Southerns	Haskel	hs@example.com	Python for Salesforce 102	1/26/2019
7	003X66	701X05	Cancelled	Dimmock	Adah	ad@example.com	Python for Salesforce 102	1/26/2019

Nitpick / Extra Credit

- Yes! Instead of dumping merge2df to one CSV and merge4df to another CSV, we could:
 - `pandas.concat(...)` merge2df & merge4df together
 - dump the result to a single CSV

Hands-On: Handling everyone not in Salesforce

Stay in your old work (if you didn't get it, go to <https://link.stthomas.edu/sfpy201901-eventmerge2>); also open <https://link.stthomas.edu/sfpy201901-info#ex5>.

1. Do a “left” merge from “eventsdf” to “contactsdf” matching on the FIRSTNAME, LASTNAME, & EMAIL; **turn on the “indicator=True” flag**; save the result as “merge3df”.
2. Remove from “merge3df” any rows where the value in the “_merge” column is not “left_only”; ensure change persists. (We do this by building an expression that becomes a “Series” of True/False values with the same “Item Ids” that “merge3df” has as “row IDs,” then putting that expression inside “merge3df = merge3df[...]”)
3. Run the following command: `merge3df = doFakeDataLoad(merge3df)`
4. Delete columns from “merge3df” so that only the columns of “eventsdf” and “ID” remain; ensure the change persists to “merge3df”. (Note: from here on out, we’ve done this before, just merge1->merge3 & merge2->merge4.)
5. Rename the “ID” column of “merge3df” to “ContactId”; ensure “merge3df” changes.
6. Merge “merge3df” against “campaignsdf” on event name & start date; “inner” merge; save the result as “merge4df”.
7. Rename the “ID” column of “merge4df” to “CampaignId”; ensure “merge4df” changes.
8. Rename the “Attendance Status” column of “merge4df” to “CampaignMemberStatus”; ensure “merge4df” changes.
9. Re-order the fields of “merge4df” to be: ContactId, CampaignId, CampaignMemberStatus, Last, First, Email, Event Name, Event Date. Don’t bother including “NAME” or “HAPPENED_ON__C” in your final output if they exist.
10. Export your data to “CampaignMemberRecordsToInsert2.csv” & have a look. Does it look like it should?

Hands-On: One Possible Answer

1. Did you get the right output?
2. Questions? (Chat room?)

```
eventsdf = pandas.concat([evdf1, evdf2, evdf3])
merge3df = eventsdf.merge(contactsdf, how='left', indicator=True,
                          left_on=['First', 'Last', 'Email'], right_on=['FIRSTNAME', 'LASTNAME', 'EMAIL'])
notInSFSeries = merge3df['_merge'] == 'left_only'
merge3df = merge3df[notInSFSeries]
merge3df = doFakeDataLoad(merge3df)

merge3df = merge3df[list(eventsdf.columns) + ['ID']]
merge3df = merge3df.rename(columns={'ID': 'ContactId'})
merge4df = merge3df.merge(campaignsdf, how='inner',
                          left_on=['Event Name', 'Event Date'], right_on=['NAME', 'HAPPENED_ON__C'])
merge4df = merge4df.rename(
    columns={'ID': 'CampaignId', 'Attendance Status': 'CampaignMemberStatus'})
merge4df = merge4df[['ContactId', 'CampaignId', 'CampaignMemberStatus', 'Last', 'First',
                    'Email', 'Event Name', 'Event Date']]
merge4df.to_csv('CampaignMemberRecordsToInsert2.csv', index=False, quoting=1)
```

LEAVE YOUR WORK UP!

Questions? (Chat room?)

Cell Intersections

Selectively Editing

Cell-Intersection Selective Editing

```
1. df['SomeColumn'][someTrueFalseSeriesSameIDs] = someValueOrSeries
```

What we're doing here is selecting a column, THEN sub-selecting certain "items" from that column the way we'd normally select rows from a table, THEN setting those cells.

You can only do this 1 column at a time, and `SomeColumn` has to exist ... sorry!

Hands-On: Adding a “notes” column

Stay in your old work (if you didn't get it, go to <https://link.stthomas.edu/sfpy201901-eventnotes>); also open <https://link.stthomas.edu/sfpy201901-info#ex6>.

1. Make a clean copy of “eventsdf” into a new DataFrame called “notesdf” (note: you'll need “eventsdf.copy()”)
2. Overwrite the contents of the “Event Name” column of “notesdf” to replace “Python for Salesforce ” with “PySF” for better skimmability. (Note: all text-filled “Series” have a “.str.replace(thingToReplace, replaceItWith)” operation.)
3. Get rid of the “Email” & “Attendance Status” columns. They're just wasting screen space right now.
4. Add a new blank column called “Note” to notesdf (add a new column & fill it all the way down as the value None).
5. Selectively edit the value of Note to say “Flag A: ” along with the Event Date from that row if the person's last name starts with a capital S.
6. Selectively edit the value of Note to say “Flag B: ” along with an upper-cased version of the person's first name if they're on the roster for an event in November 2018 or later.
7. Verify “notesdf” now looks something like this: *(HEY! Why is “Southernns” “flag B” in 2 rows?)*

	A	B	C	D	E
1	First	Last	Event Name	Event Date	Note
2	Revkah	Lilburn	PySF101	10/20/2018	
3	Haskel	Southernns	PySF101	10/20/2018	Flag A: 2018-10-20
4	Ermanno	Withinshaw	PySF101	10/20/2018	
5	Haskel	Southernns	PySF101-Office Hours	11/10/2018	Flag B: HASKEL
6	Julianna	Judron	PySF102	1/26/2019	Flag B: JULIANNA
7	Haskel	Southernns	PySF102	1/26/2019	Flag B: HASKEL
8	Adah	Dimmock	PySF102	1/26/2019	Flag B: ADAH

Hands-On: One Possible Answer

1. Did you get the right output?
2. Questions? (Chat room?)

```
notesdf = eventsdf.copy()
notesdf['Event Name'] = notesdf['Event Name'].str.replace(
    'Python for Salesforce ', 'PySF')
notesdf = notesdf.drop(columns=['Email', 'Attendance Status'])
notesdf['Note'] = None
conditionAseries = notesdf['Last'].str.startswith('S')
notesdf['Note'][conditionAseries] = 'Flag A: ' + notesdf['Event Date']
conditionBseries = notesdf['Event Date'] > '2018-10-31'
notesdf['Note'][conditionBseries] = 'Flag B: ' + notesdf['First'].str.upper()
print(notesdf)
```

Questions? (Chat room?)

Pivoting Data

This is such a huge topic. We'll do one.

See <http://pbpython.com/archives.html> and start towards the end (2014) if you need lots of this.
(**“Practical Business Python”** blog)

Door Prize – <https://link.stthomas.edu/sfpy201901-info#pivot>

“Door Prize Script: A little pivot”

	A	B	C	D	E	F	G	H	I
1	First	Last	Email	10/20/2018	11/10/2018	1/26/2019	RSVPed	Came	Didnt
2	Adah	Dimmock	ad@example.com			Cancelled	1	0	1
3	Ermanno	Withinshaw	ew@example.com	Attended			1	1	0
4	Haskel	Southerns	hs@example.com	No-Show	No-Show	Attended	3	1	2
5	Julianna	Judron	jj@example.com			No-Show	1	0	1
6	Revkah	Lilburn	rl@example.com	Attended			1	1	0

```
import numpy
import pandas

evdf1 = pandas.read_csv('https://raw.githubusercontent.com/pypancsv/pypancsv/master/docs/_data/mergehandson_event1.csv')
evdf2 = pandas.read_csv('https://raw.githubusercontent.com/pypancsv/pypancsv/master/docs/_data/mergehandson_event2.csv')
evdf3 = pandas.read_csv('https://raw.githubusercontent.com/pypancsv/pypancsv/master/docs/_data/mergehandson_event3.csv')
eventsdf = pandas.concat([evdf1, evdf2, evdf3])

pivotdf = pandas.pivot_table(eventsdf, index=['First','Last','Email'], columns='Event Date',
                              values='Attendance Status', aggfunc=numpy.min)
pivotdf = pivotdf.reset_index()
pivotdf.columns.name = None
eventDatesOffered = list(eventsdf['Event Date'].unique())
pivotdf['RSVPed'] = pivotdf[eventDatesOffered].count(axis='columns')
pivotdf['Came'] = pivotdf[eventDatesOffered].isin(['Attended']).sum(axis='columns')
pivotdf['Didnt'] = pivotdf[eventDatesOffered].isin(['No-Show','Cancelled']).sum(axis='columns')
pivotdf.to_csv('c:\\example\\outputpivot.csv', index=False)
```


Questions? (Chat room?)

THANK YOU! - 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)

