Python Programming Concepts

with instructor Colby Witherup Wood

Workshop originally created by Christina Maimone

This workshop is brought to you by

Northwestern IT Research Computing and Data Services

Have a programming or data question about your research?

We're here to help. **bit.ly/rcsconsult**

This workshop was designed for anyone who is about to start learning a coding language.

Goals:

Build familiarity with:

1. How to give computers instructions
2. Common terms and concepts

How to ask questions

Just ask!

The bathrooms are…
The emergency exits are…

Downloading materials from GitHub

https://github.com/agithasnoname/programmingConcepts

Click on the green **Code** button

then click on **Download ZIP**

GitHub makes it difficult to download single files, so you should generally always download the entire Repo (repository).

Programming languages

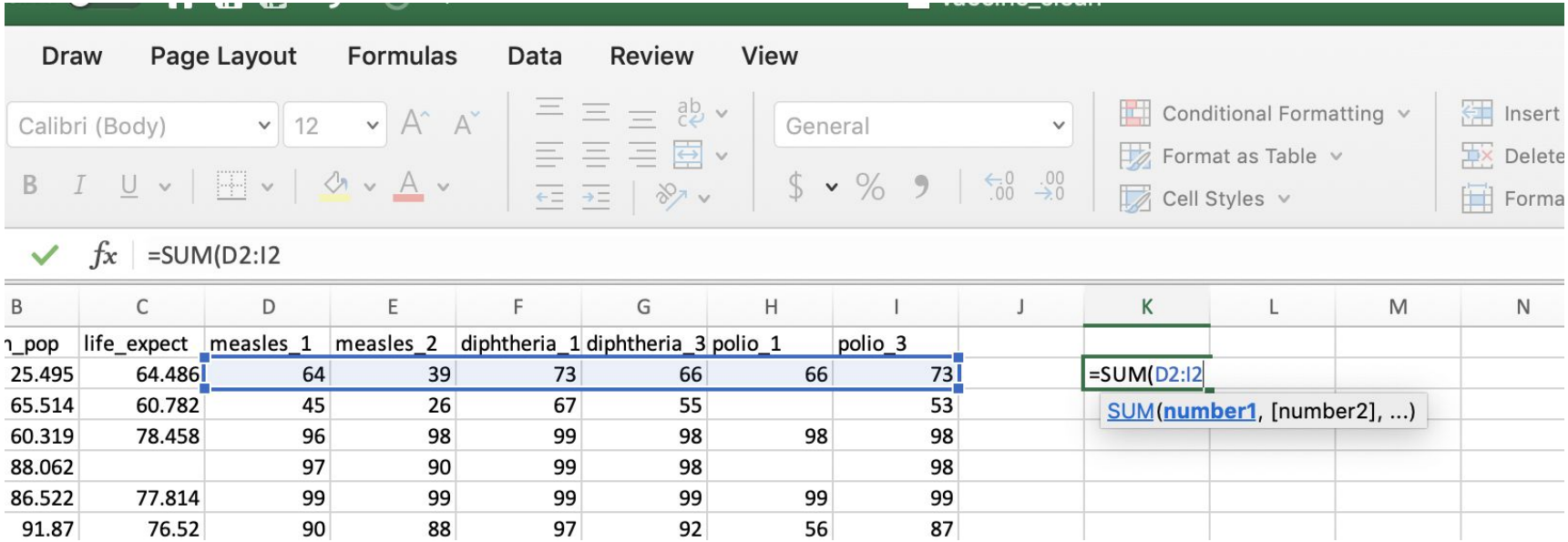
How you talk to your computer

Modern computers can interpret many different languages

GUIs (graphical user interfaces) allow you to talk to your computer without knowing any programming language

Programming languages

requires you to use specific words or characters in a specific order



B	C	D	E	F	G	H	I	J	K	L	M	N
_pop	life_expect	measles_1	measles_2	diphtheria_1	diphtheria_3	polio_1	polio_3					
25.495	64.486	64	39	73	66	66	73		=SUM(D2:I2			
65.514	60.782	45	26	67	55		53					
60.319	78.458	96	98	99	98	98	98					
88.062		97	90	99	98		98					
86.522	77.814	99	99	99	99	99	99					
91.87	76.52	90	88	97	92	56	87					

Programming languages

The **command line** is how we can talk directly to our computer without a GUI.

Different computers have different **shells** to access the command line and different languages you use on the command line.

Mac: Terminal uses Unix Bash or zsh, PC: Windows PowerShell

These are designed for controlling your operating system and computer: installing programs, moving files, etc.

How do we talk to our computer in Python or R?

- **Interactive programming** - through a shell, one line at a time

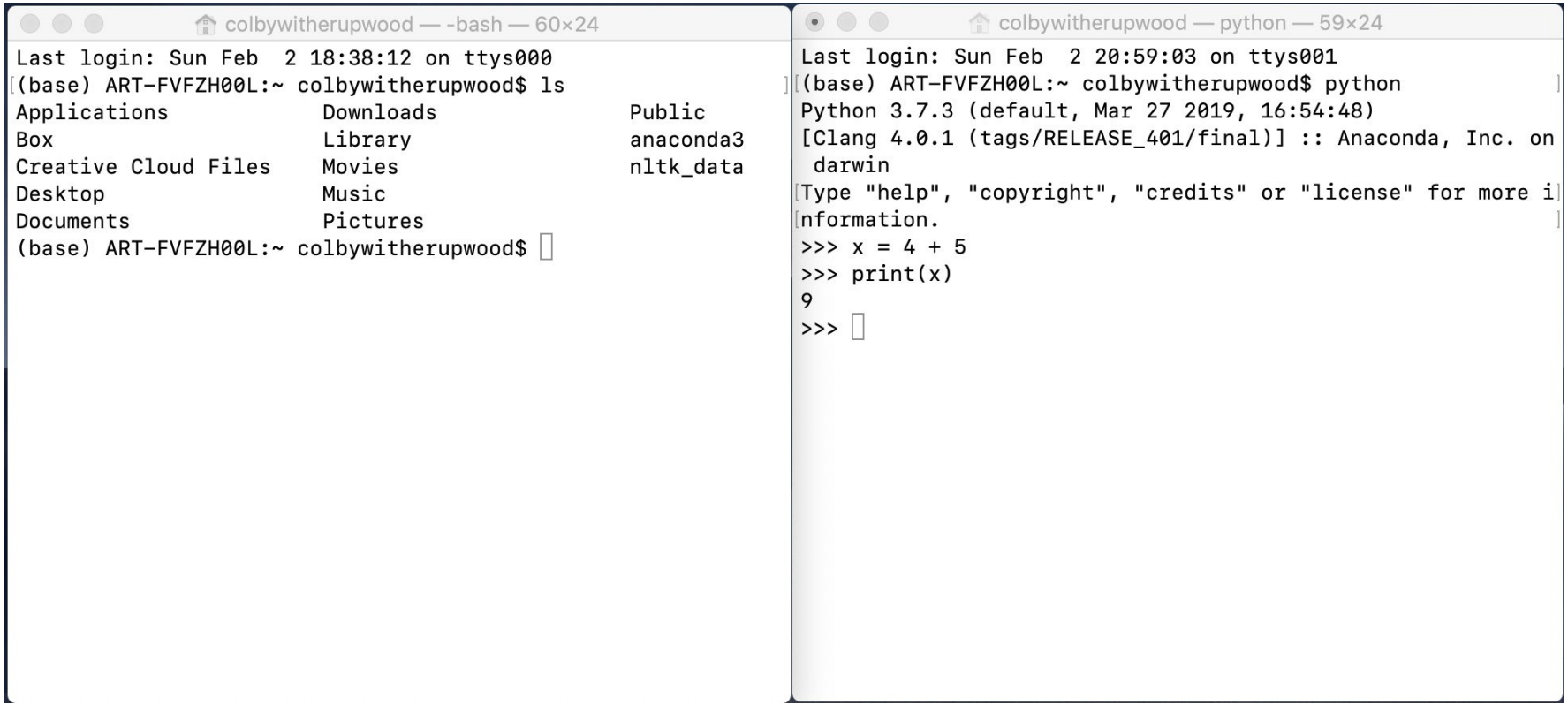
- **Batch programming** - running a whole **script** (a plain text file that contains one to many lines of code)

- Coding notebooks allow you to run code in chunks and view output directly below.

How do we talk to our computer in Python or R?

- With the help of a GUI. GUIs for coding are called **IDEs - Integrated Development Environments.** They allow both interactive and batch programming.

Command prompts



Time to Review!

Open the ProgrammingConceptsReview document from the folder you downloaded.

Filesystems

Filesystems

Coding requires us to move away from point and click.

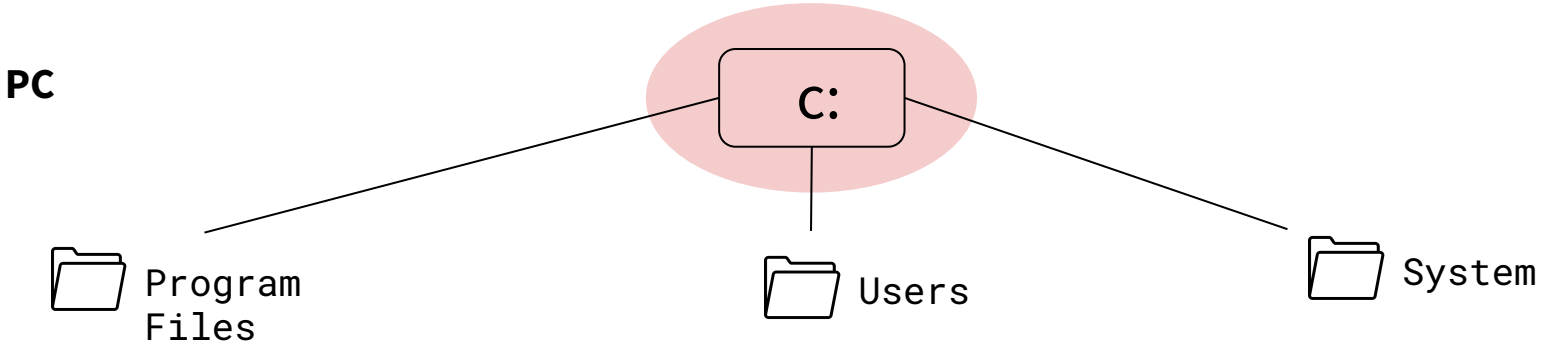
We will want to work with files, so we need to know how to use words to guide the computer to the right files.

Every file has an **absolute path**, which starts with the **root**.

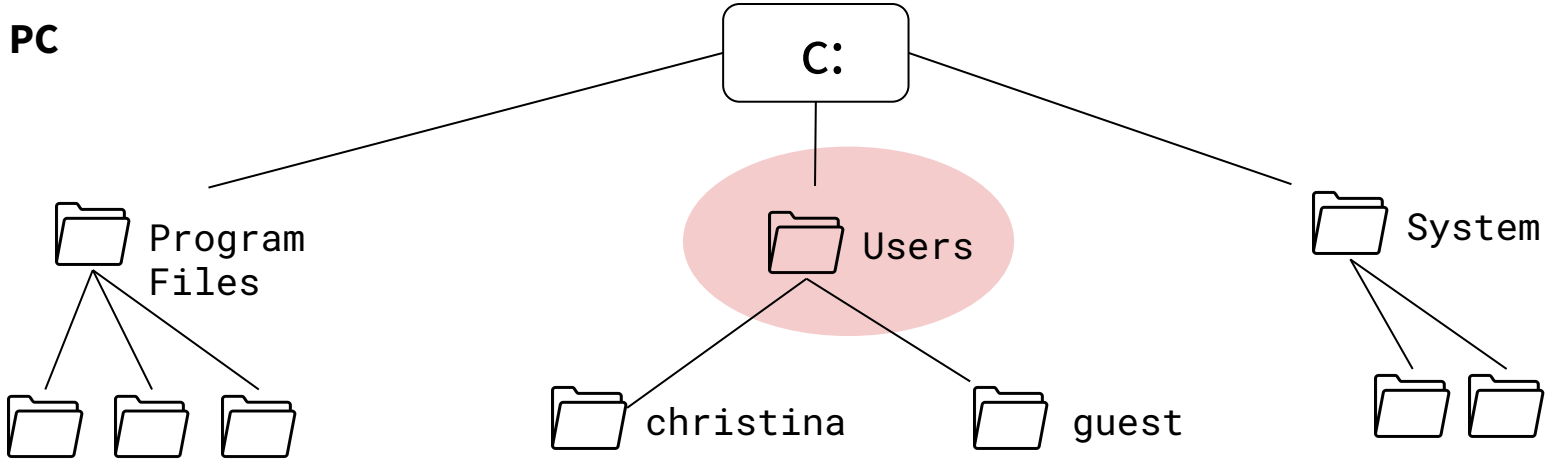
PC

C:

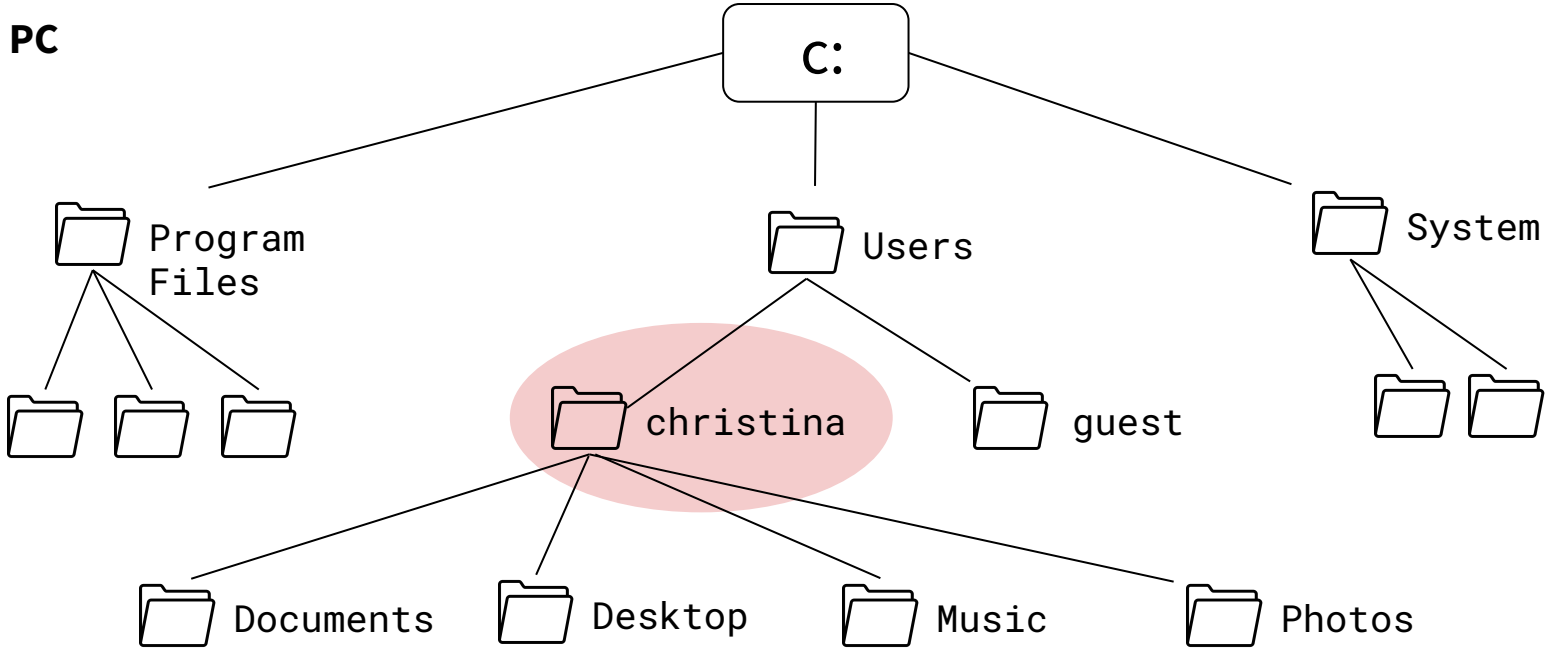
C:\



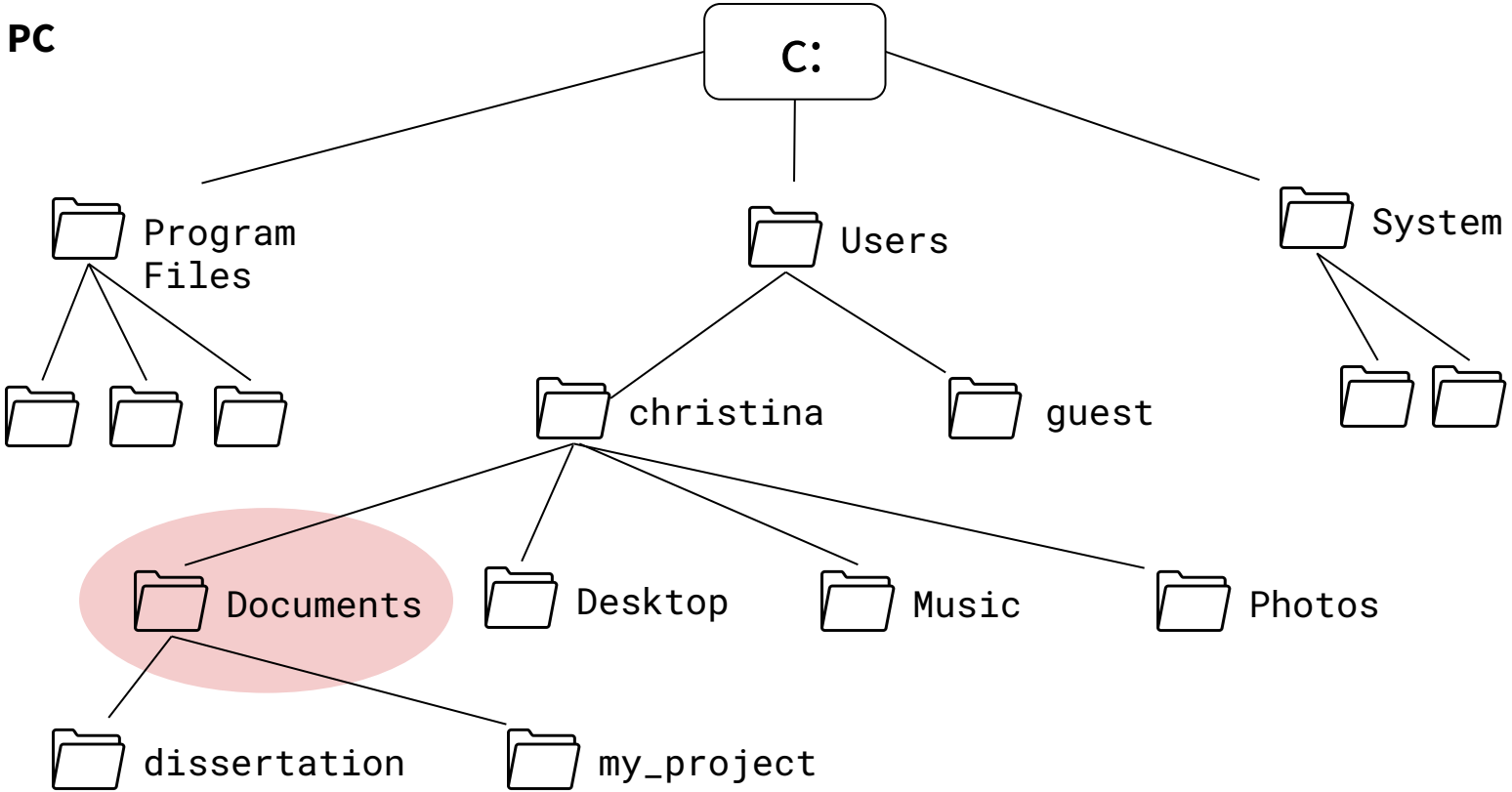
C:\Users\



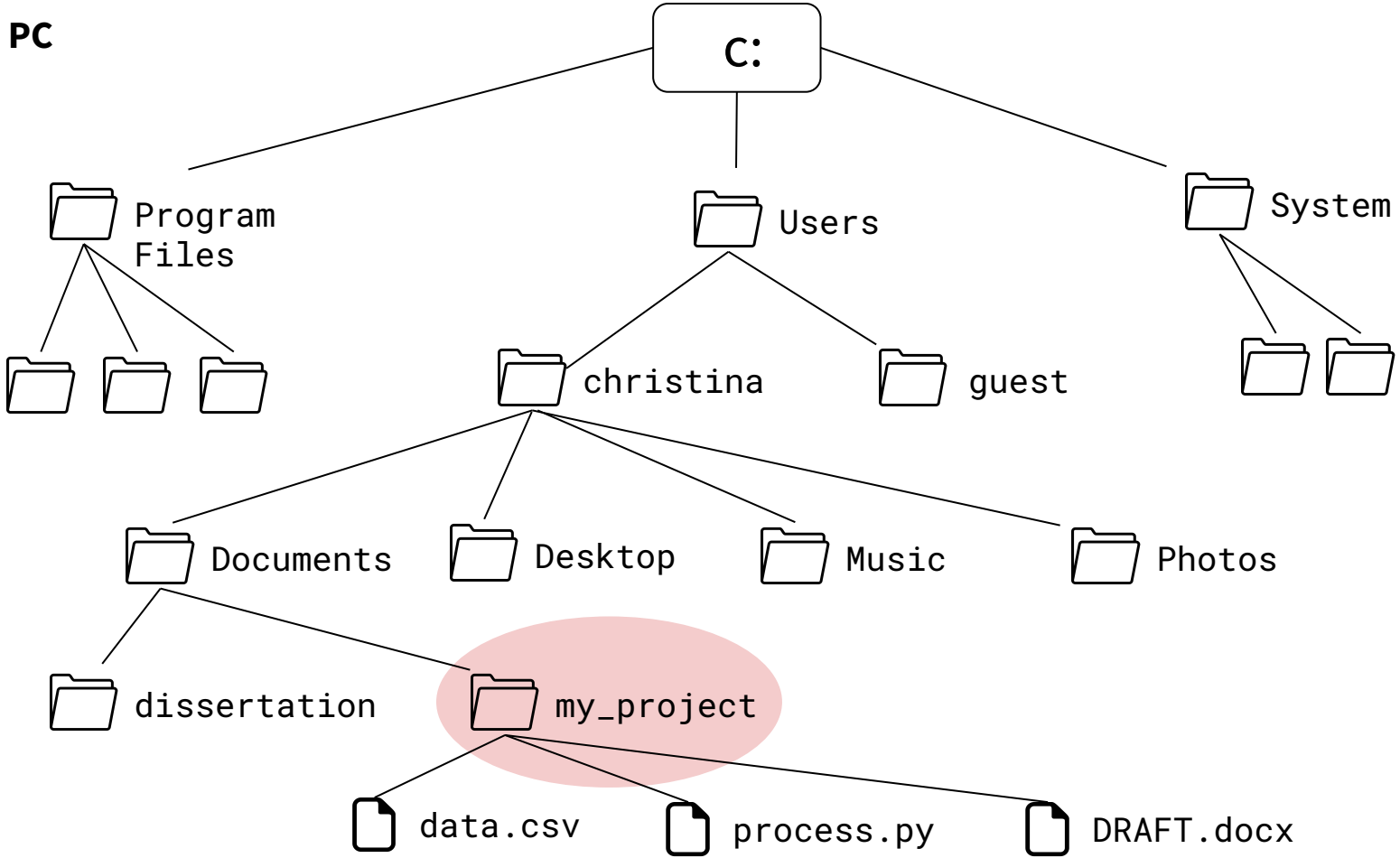
C:\Users\christina



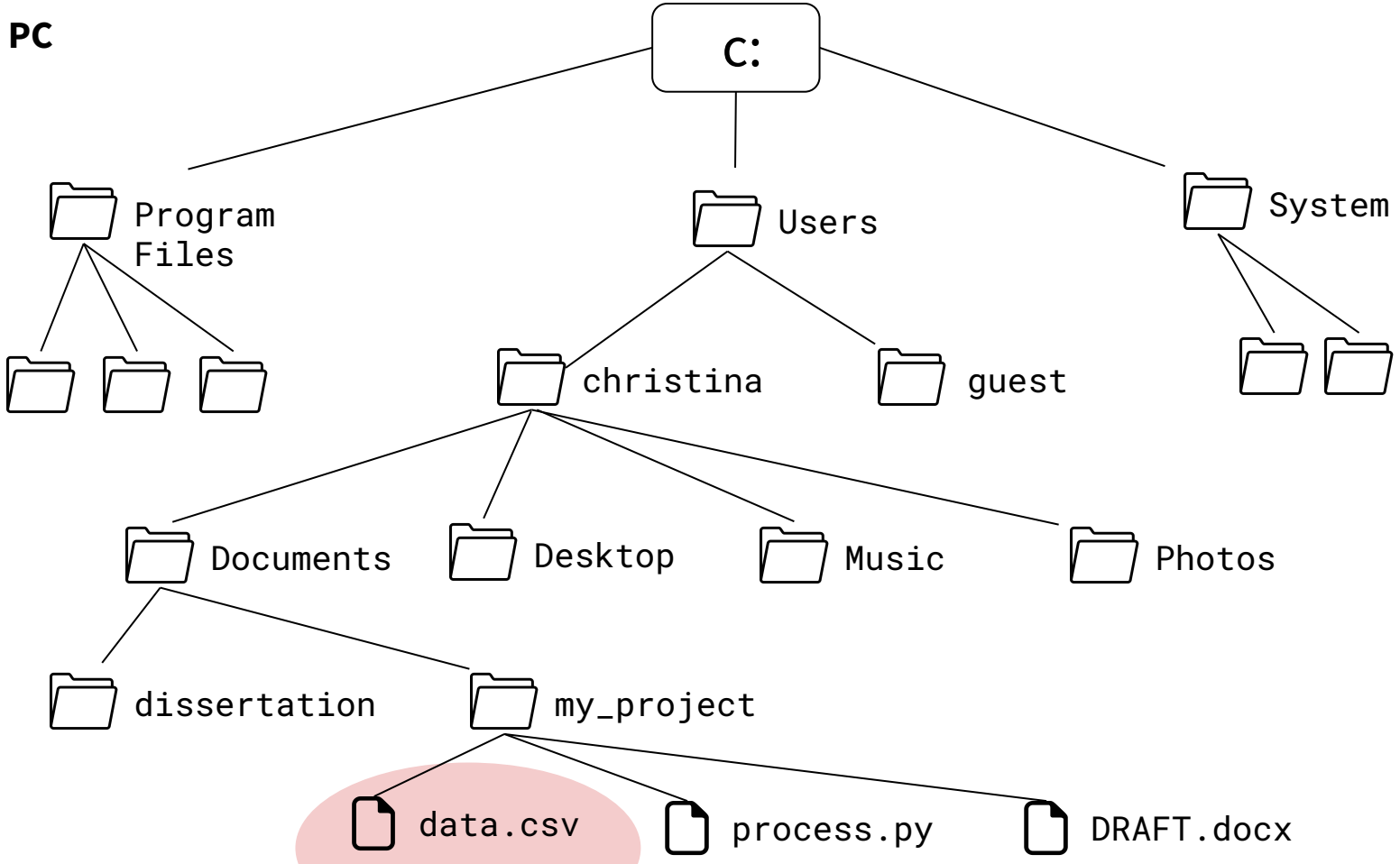
C:\Users\christina\Documents



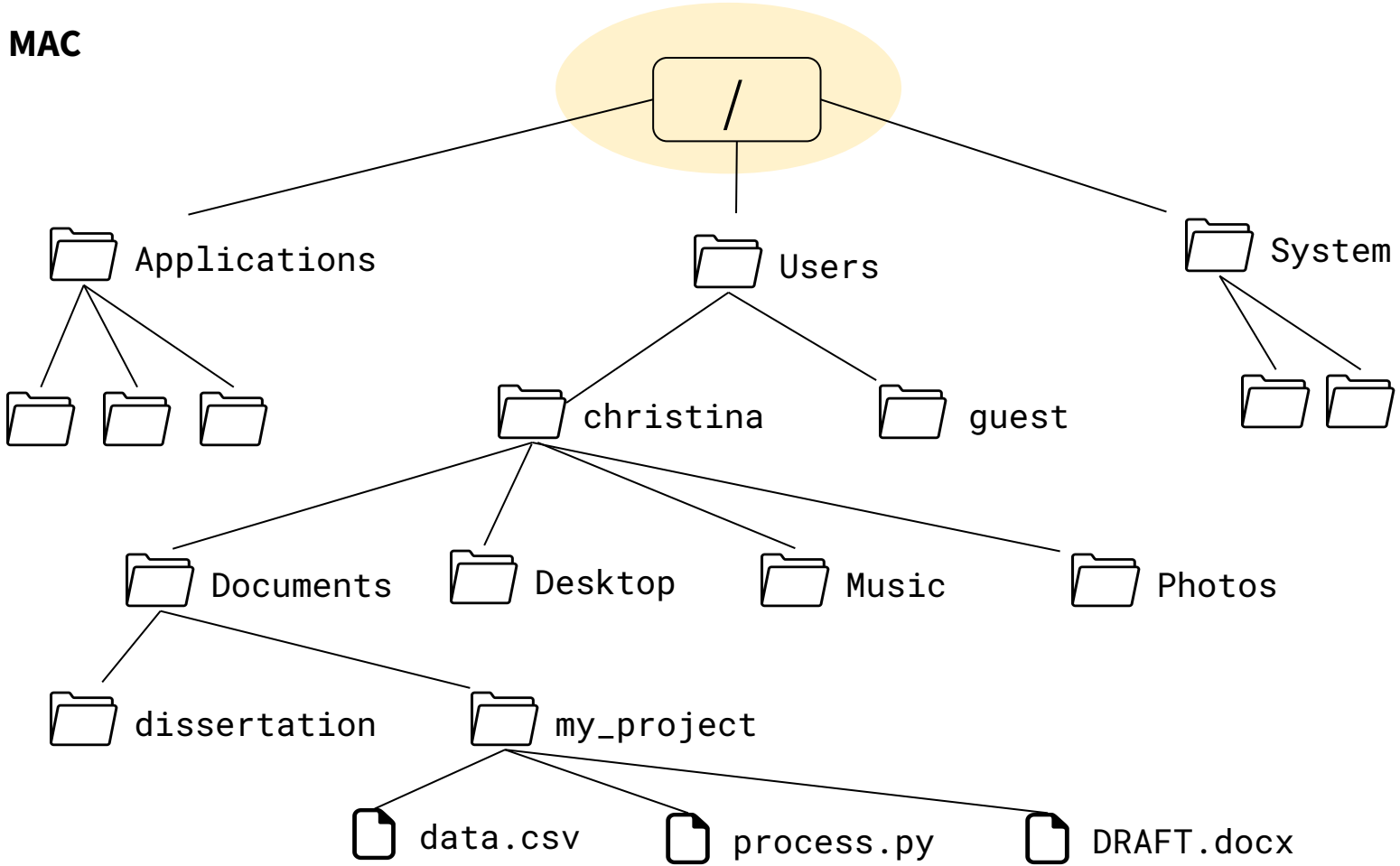
C:\Users\christina\Documents\my_project



C:\Users\christina\Documents\my_project\data.csv



/Users/christina/Documents/my_project/data.csv



Absolute Paths

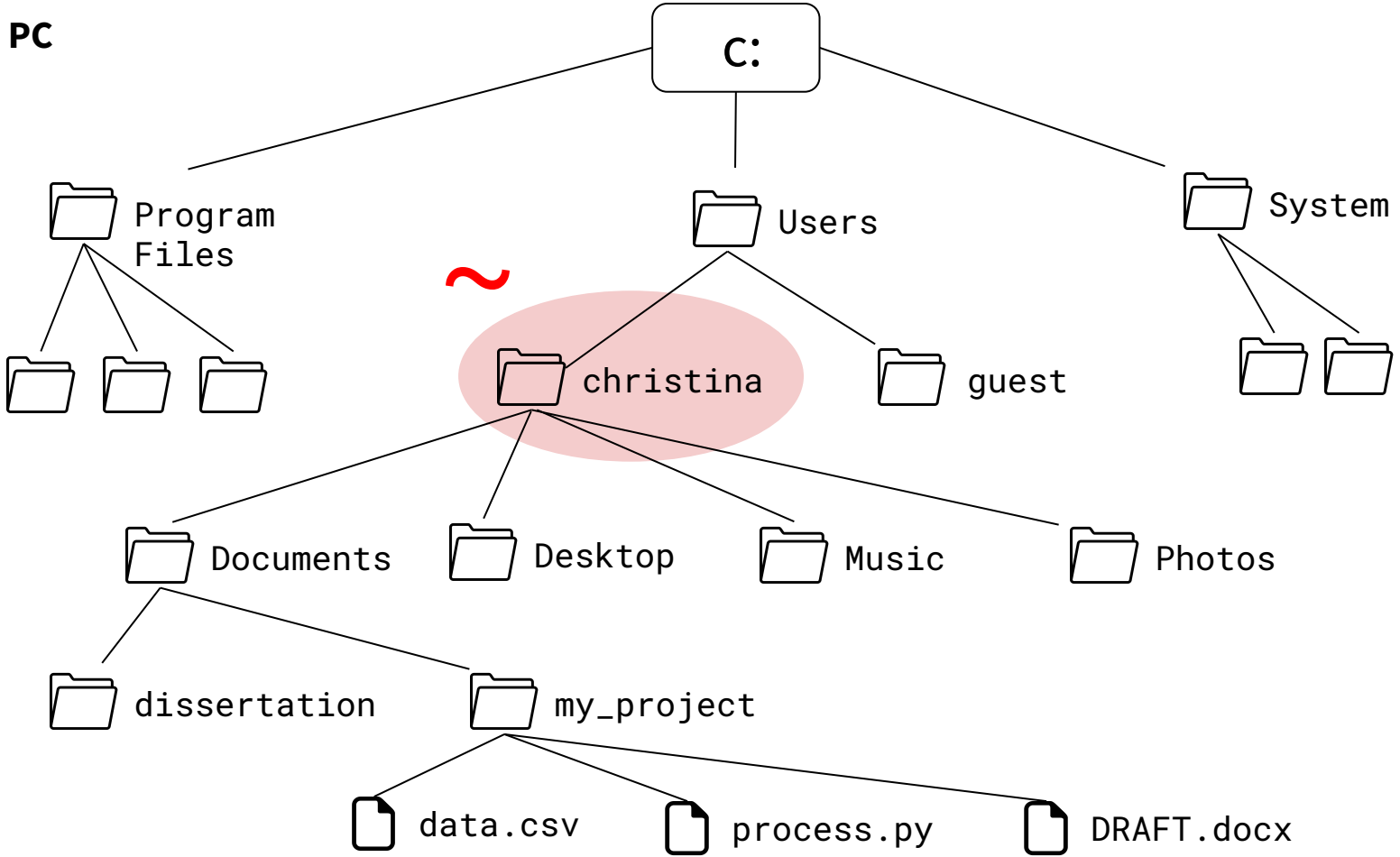
```
C:\Users\christina\Documents\my_project\data.csv
```

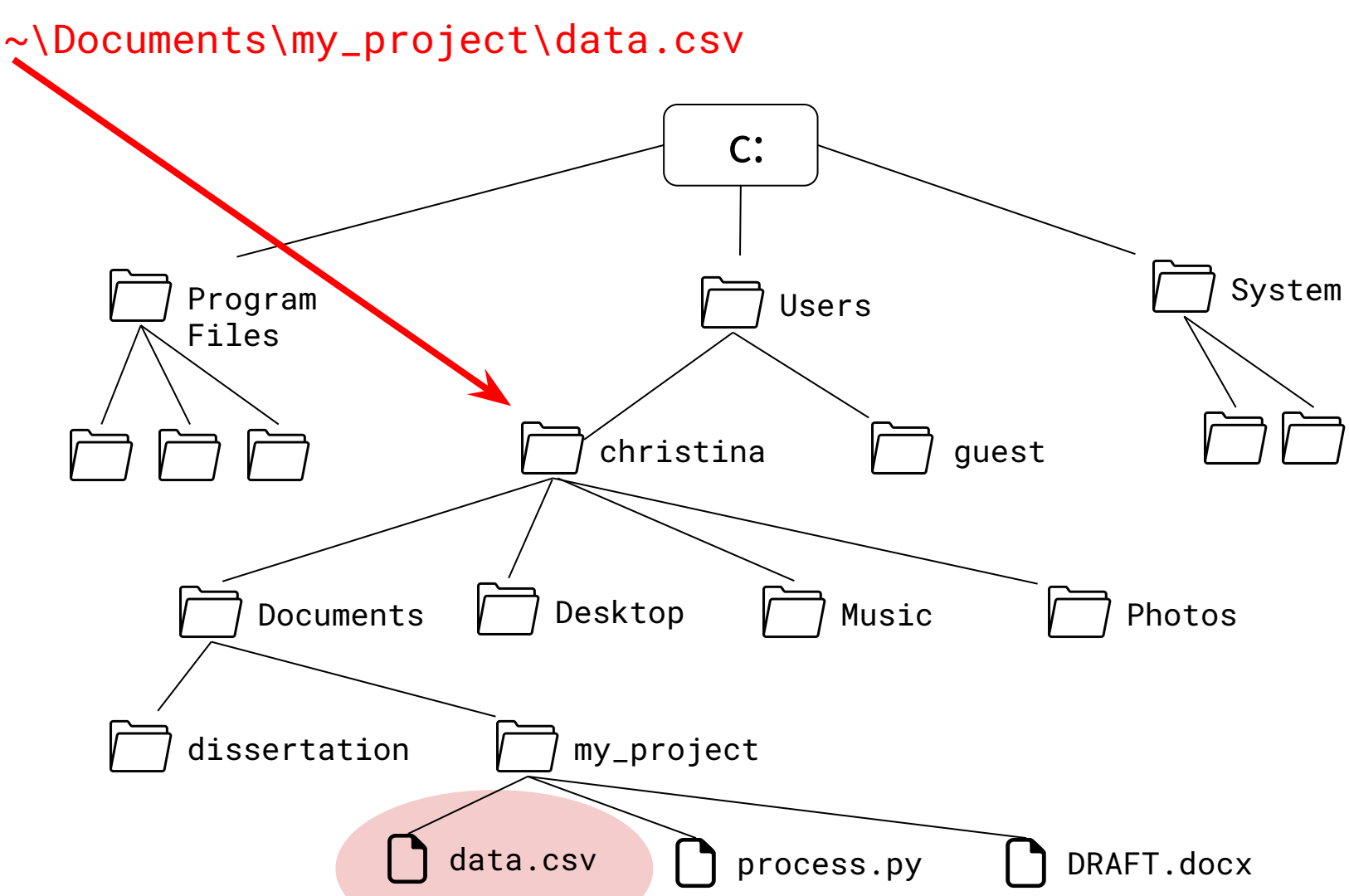
```
/Users/christina/Documents/my_project/data.csv
```

Home directory

In addition to a root directory, computers have a **home directory**. As a shortcut, you can refer to the home directory as ~

Home Directory: C:\Users\christina = ~




~\Documents\my_project\data.csv
C:
Program Files
Users
System
christina
guest
Documents
Desktop
Music
Photos
dissertation
my_project
data.csv
process.py
DRAFT.docx

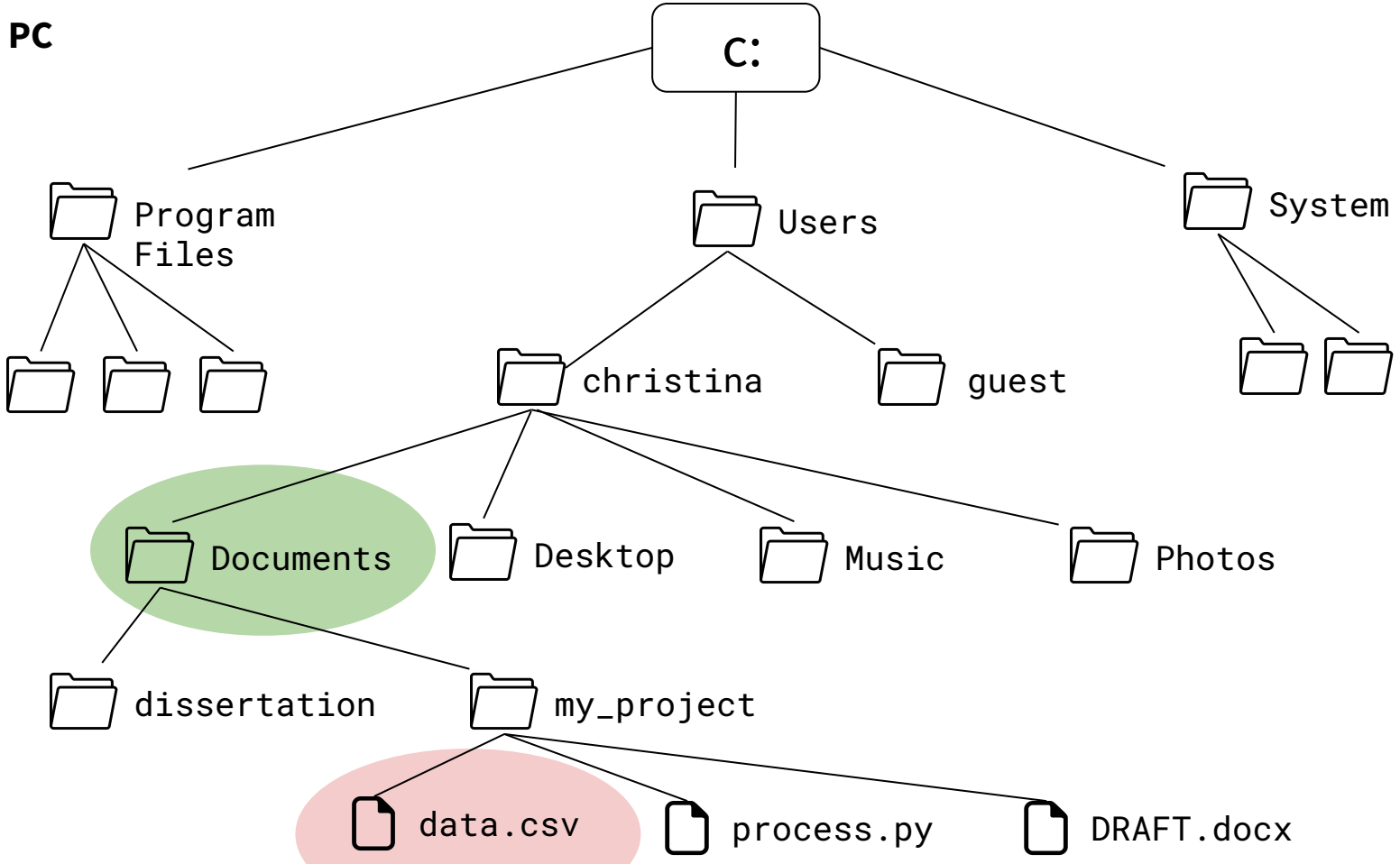
Working Directory

A **working directory** is the directory associated with a running process or program

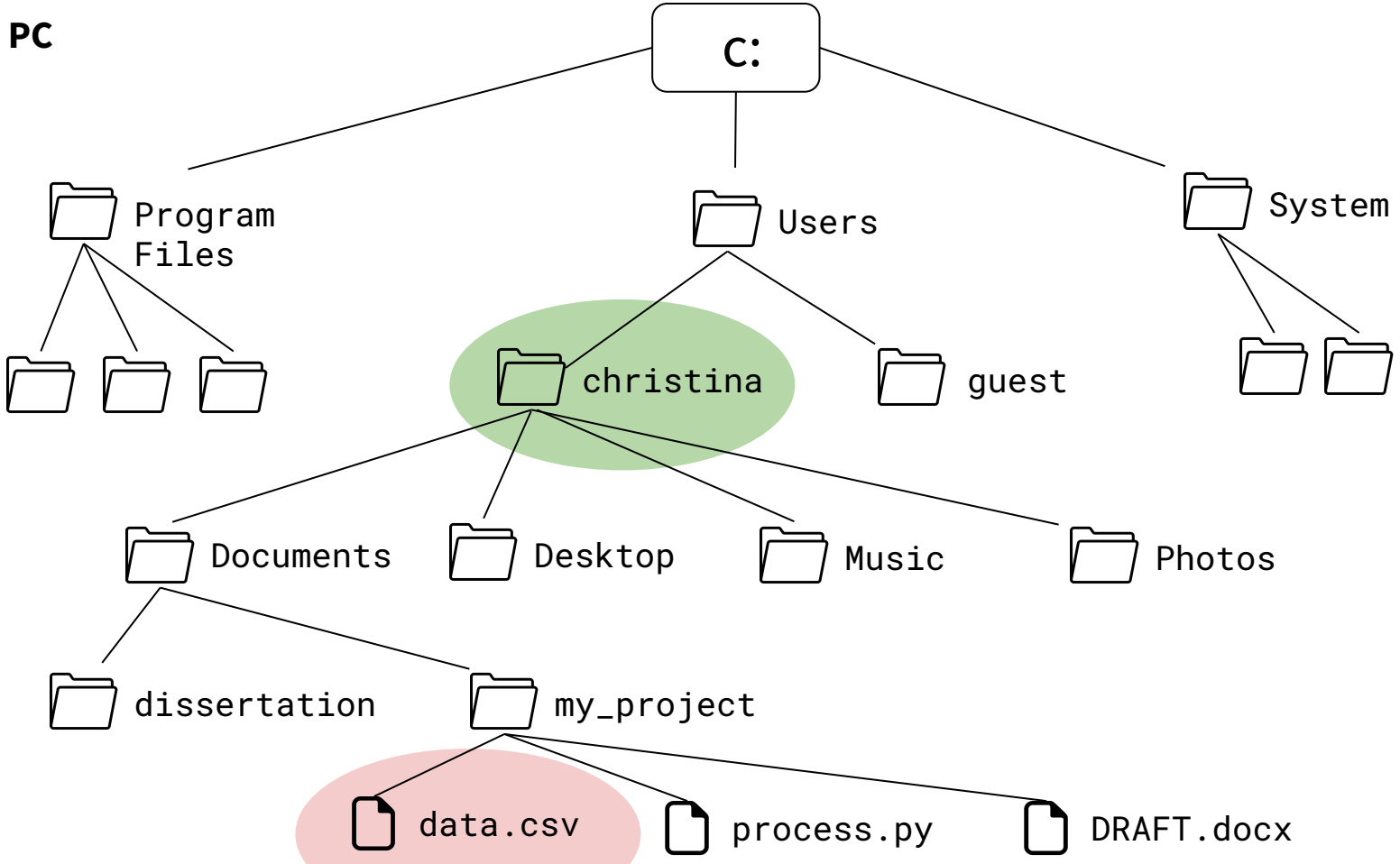
This is where the computer starts when looking for files

You can use **relative file paths** from your working directory

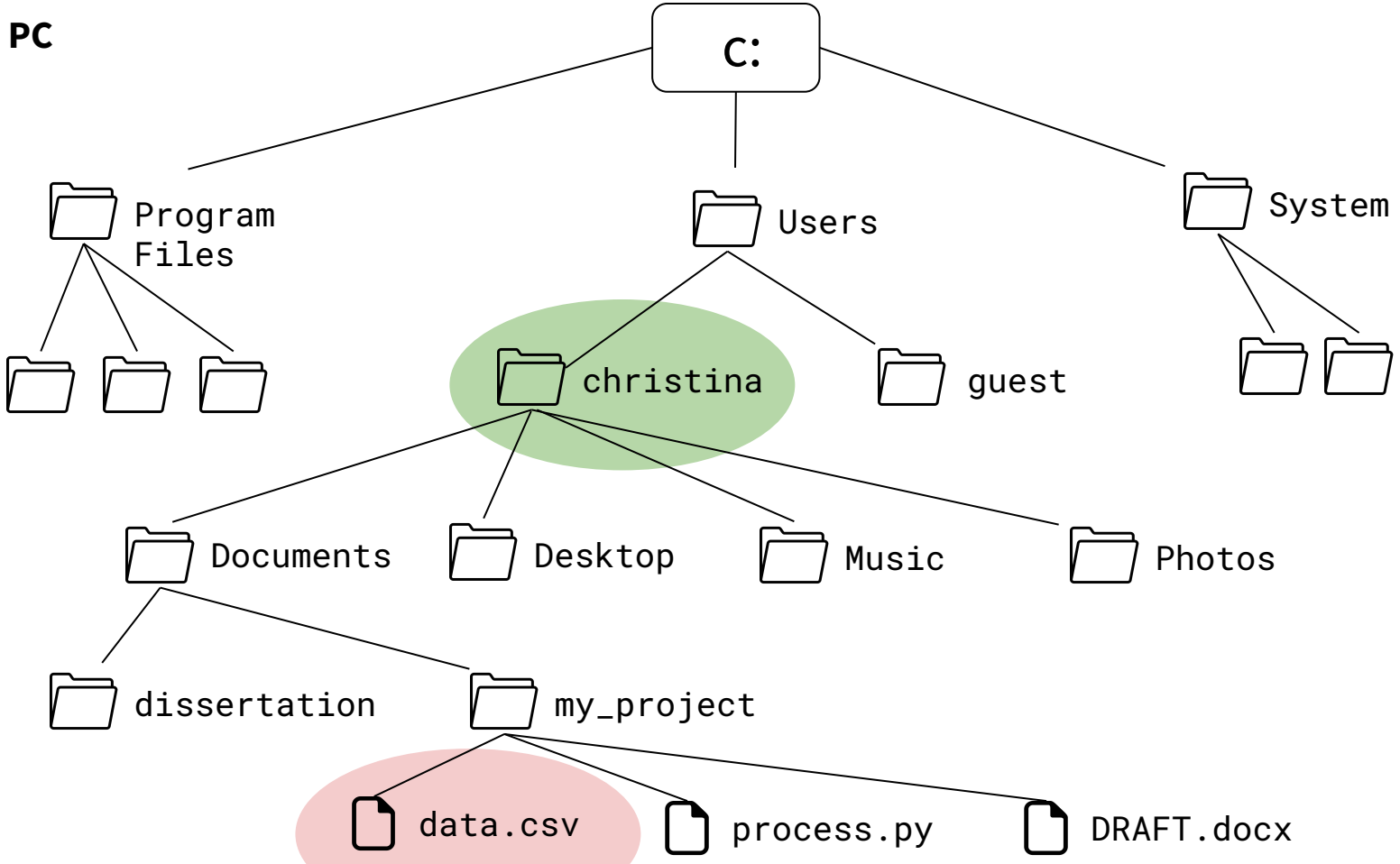
Relative Path from Documents: my_project/data.csv



From christina:

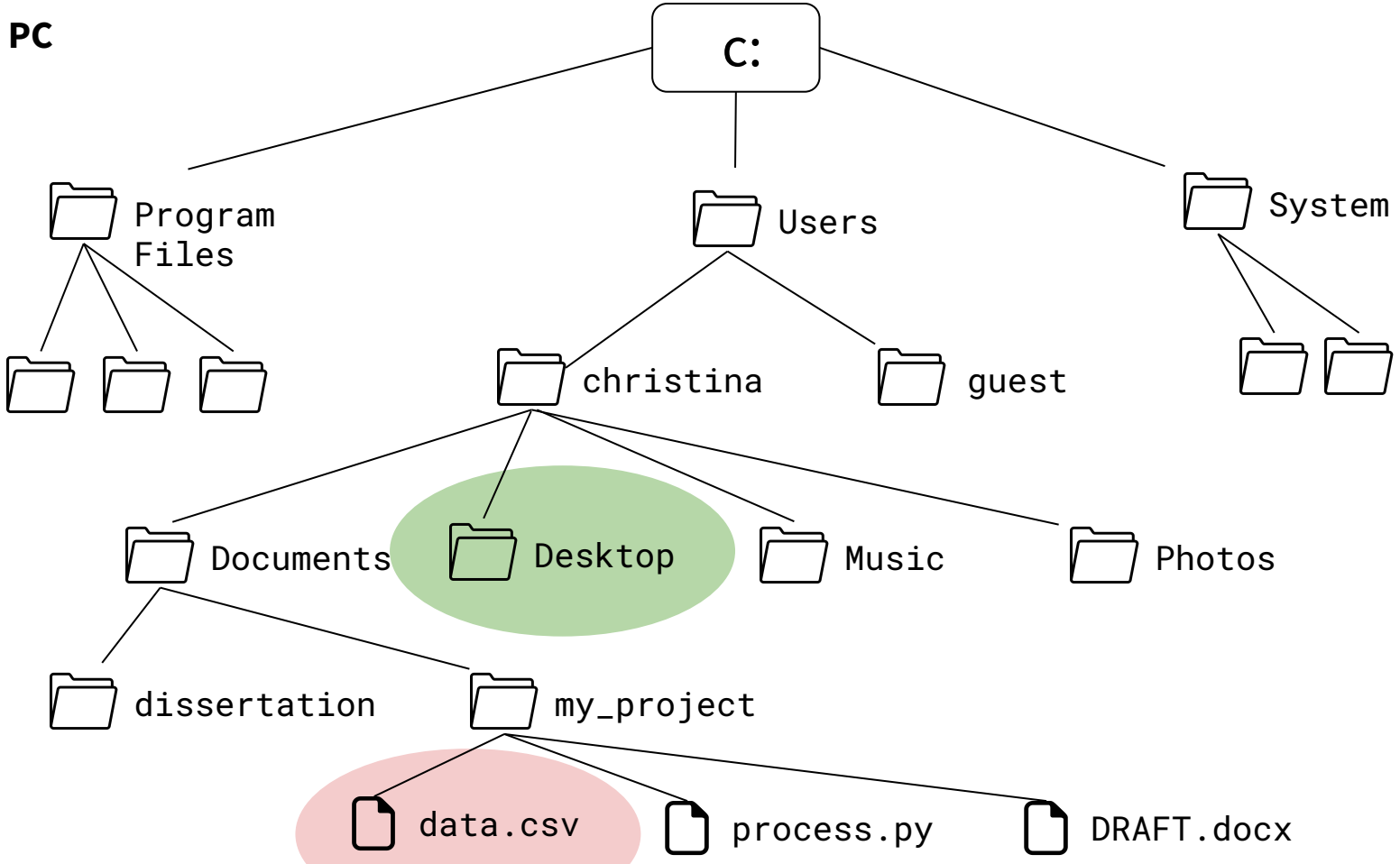


From christina: Documents/my_project/data.csv

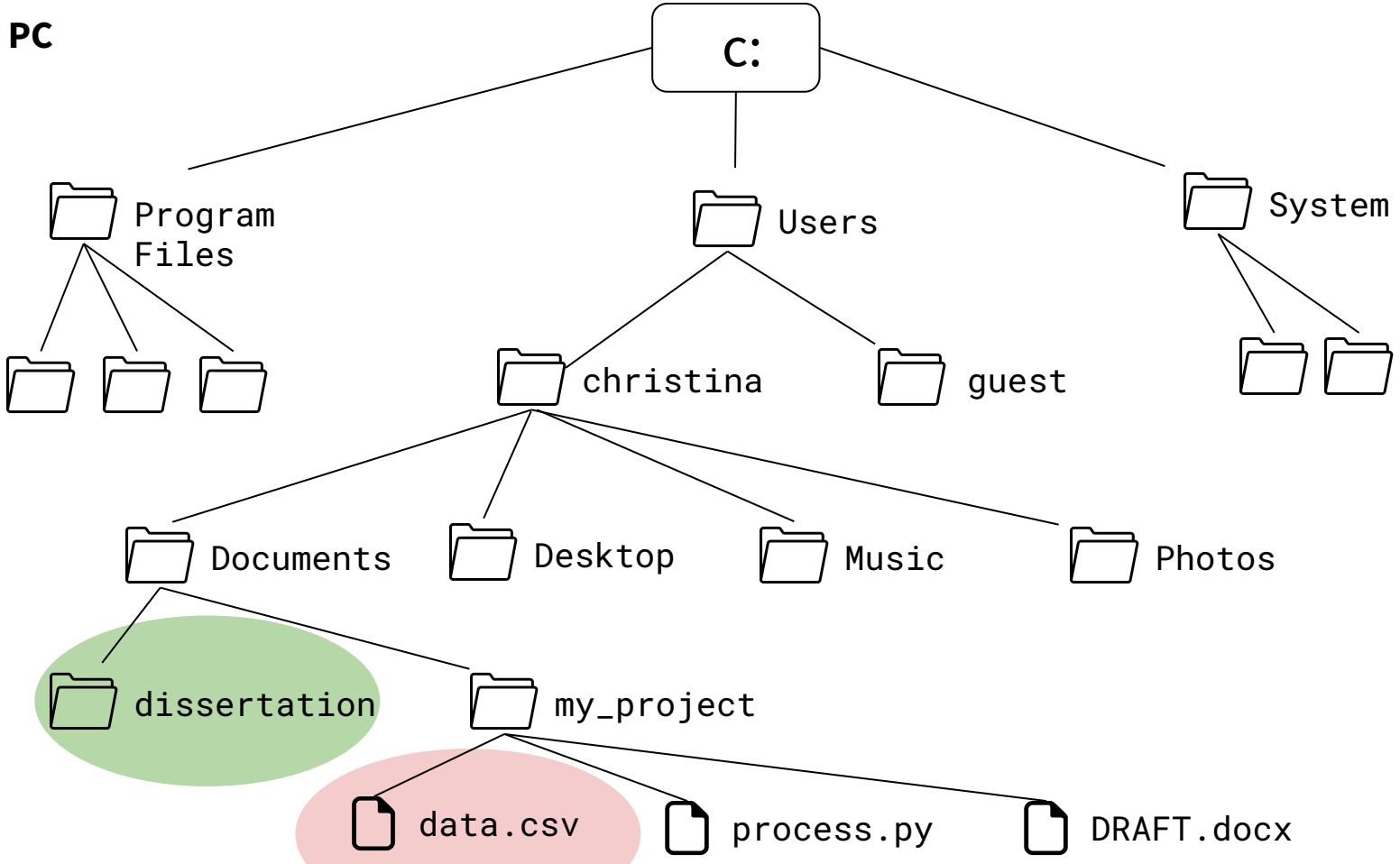


Another shortcut is .. which goes up one directory.

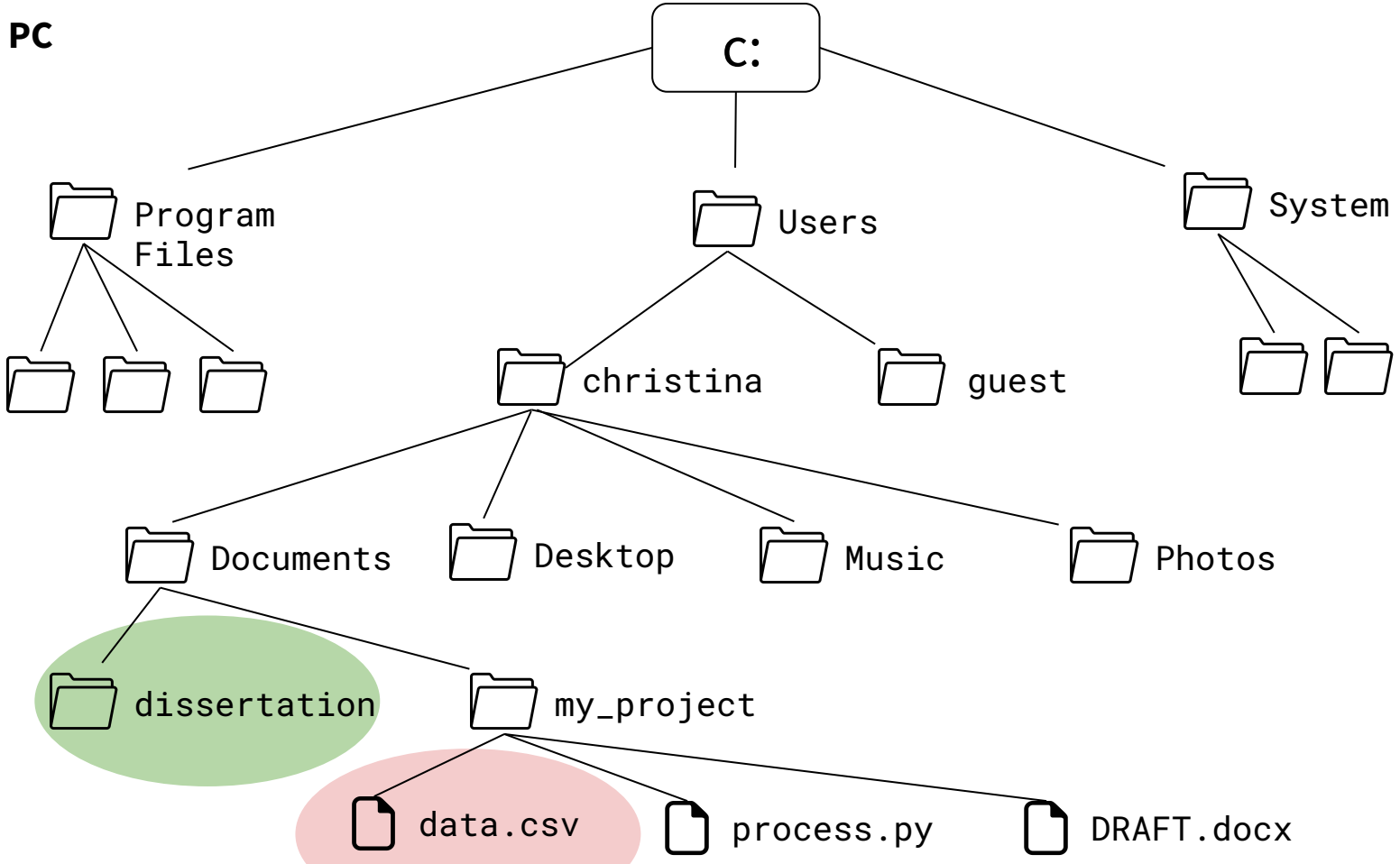
From Desktop: ../Documents/my_project/data.csv



From dissertation:



From dissertation: ../my_project/data.csv



But what is the default Working Directory?

Python:

- **Where you start Python from**
- **Where you call a Python script from**
- **Where your notebook is saved**

You can always set or change the working directory

Store Project Files Together

project_directory

- code
  - process_data.py
  - make_graphs.ipynb
  - estimate_model.py
- data
  - people.csv
  - exam1.csv
- results
  - impact_plot.pdf
- reports
  - chapter1.docx

Why this matters

When writing scripts, you'll want to think about how you include your file paths. Where will you be in your file system when you run the script? Will you always be running it from the same place? Will the script break if you ever reorganize your project directory?

Bonus tip

Stop using spaces in your folder and file names. Mac and Linux machines don't read spaces in file names the same way as Windows. Use an underscore_instead. Or camelCase. This will make your code more portable in case you need a friend to test it or you want to run it on a server like Quest.

Files

Reading and Writing

Read: Open a file to get the contents

Write: Open a file to put information in

Modes

Read: get information, can't change it

Write: empties the file! then allows writing

Append: add to the bottom of the file

File Types

Text: Restricted set of characters

>> Data can be opened and viewed directly

Binary: Custom data

>> Needs a program to interpret and display the data

Plain Text Files

NO FORMATTING
NO IMAGES

Common extensions: .txt, .tab, .csv

Also plain text:

- Data files: XML, JSON
- Markup: HTML, Markdown (.md), LaTeX (.tex)
- Code: R scripts (.r), Python scripts (.py)

Plain Text Files

The extension doesn't determine if it's a plain text file. The content does.

Common extensions: .txt, .tab, .csv

Also plain text:

- Data files: XML, JSON
- Markup: HTML, Markdown (.md), LaTeX (.tex)
- Code: R scripts (.r), Python scripts (.py)

```
# R Workshops

This repository is a clearing house for resources for individual R workshops from [Resea

# Workshops

## Current Workshops


[Intro to R](https://github.com/nuitrcs/r_intro_june2018)

[`ggplot2`](https://github.com/nuitrcs/r_ggplot_july2018)

[Databases](https://github.com/nuitrcs/databases_workshop/tree/master/r): Information on
useful reference, but you'll need a database connection to run it.  See that repository f

[R Markdown](https://github.com/nuitrcs/rmarkdown_workshop)

[R Shiny](https://github.com/nuitrcs/rshiny)


# Software

For workshops, it's best to install R and RStudio on your own laptop (both are free).

[Install R](https://cran.rstudio.com/)

[Install RStudio Desktop](https://www.rstudio.com/products/rstudio/download/)
```

https://raw.githubusercontent.com/nuitrcs/rworkshops/master/README.md

```
COMPND      Ammonia
AUTHOR      DAVE WOODCOCK  97 10 31
ATOM      1  N            1       0.257  -0.363   0.000
ATOM      2  H            1       0.257   0.727   0.000
ATOM      3  H            1       0.771  -0.727   0.890
ATOM      4  H            1       0.771  -0.727  -0.890
TER       5               1
END
```

File: ammonia.pdb

```
%PDF-1.5
%-‘≈ÿ
13 0 obj
<<
/Length 1063
/Filter /FlateDecode
>>
stream
x⁄≠VYo‹6¿~¯ØPÛD¿¿#¿¿•æ5i¿¥pÉ†›¢¿u¿-¿Ω+D+.t‘¬øÔêC≠•µú¿Õæ,Ø˘Ê˙fFØwWØfiÚ4bí
ôÛhw¿)¿)ñ“T®hWG¿ë¿¬®„D®åÏb¡»√   6åò!˛{˜À´∑,èJZÊ<w¢iîpEUY†‰¿{<⁄¿û¿AjD¿íå1/<
˛ôòë¿èõÆjß⁄¿|è»ú-ê¿ßLÅQ¬¿-Ö@¯¿Œ“¿#‡©¿¶lÒè^[Ø»∂Fw¿∏Ixù0FÀ,˚
‡ü;Á«hˆ¶˘¶∏Ô¶£ÈõÍõbæ9Ë^W#Z¿•¥,
∏Ç¨ªdÁêÏ,Öî¿/√¿y)…?qñ¿›7˙∂=g´$’|}¿ÚË7¶¿¿†ƒØ?,~xb˝M±∫°q"≥å¸y0¿Ø¿¬pß’°ˆ£©Òx¥¯?´m¸´1NòÁ¿$·•˜
çÕ~9üÚ(+¿öe¿}ª¥HÃ∆Wˆh¿<Iëå†Üë¿◊_"∆Ë¿Ÿí=F6+RÌj )˘’¿£{õìVw˚IÔQ¿B€¿fi∑å¿¥s¿=«ª!TI’Ë¿OV≈¿¿Ü©r¡8
¿°Pò\,È¿å,JQ)ãïeÔ˛Πæfi™ÅÖ/"e4/ÚïòÏ?°G0AãT≠¬ÿŒ|Nè(h*≥ïÿx-!pMW7ï¿Cl∫¿5‰g;¿G%Yí›¡”Ù´"®{≥f¿‰./
hûghC›‹§Bòfi`ÕcŒÔz{DÜ¿õa¿ùM∑«˝Ÿ$|¿Ï∂} ÀMö•ÜÓÈV˛JA%¿Á^◊√VißTÊÁ“¿∞@æÑß¿t%◊MÛ±ÙWm¿˜
}wÊü ¿!§‡fi}Ëÿzxé˘6Ä]fiH¿?˚µ¿*0¿
∑%¿V˝>ˆ¿≠≠¿í§Tdeò#%æfi≈¿òs;°Dµ√IWffiËÆ∆EgÓq—6ùAk*ÁŒÏ¿¿æKµ_ò¿¬Ò‘Œ¿lÔûsv¿≤ï¥`±è:¿YaBv[π¿T¿|Œ]3
€¥°'©ö$¿Øg (¿˘Ø"?¿¿©/EJ9¿Å7gj¿<I¿¿≠˘±E63e⁄”AáÓ8¿Á[°∂}mzflÈ¿π∂ÆW›c{¿¿÷CÄmÕ∏ë©œTflt
≈|¿'f8ØôCcà¿1¿ãŒôø(}…
 3Õ˛_¢ü˘÷b±ƒ\=Ø9ÿo¿DWœl¿ÒÇiâãSo¯Ω>ûßH°¿Ê§@0.ü§#'Gg¡‰∫]ËêãaÎ∆Bfl€©´qÑ∫K◊¯∞i~xÊÁCï∫o¿/Ú[hè]
Äx˘0∫ ⁄Œ¿ü"¿ÿ“¿Ñƒ¶êi⁄NÓK·íV8èEQPëóOÁÒE›1¿æ0¿_8¿ß”`jßú)r;ç¯¿¢¯Ä∑¿¿¿¿∑”#¿Øü¶
í°o¿v¿Ú¿¿m¿¿¿k¿¿^>¥aã.¿»B;w¿'C3å¿?>æ˚ƒ¿Ñ°   ◊Yä¿}}1L∑/6Ìíó T*Ò%u∫
Î•N¿CCŒ¿N”‡Y≤•/S4¿Á∑ûoW?ÌÆ˛¿¥¿
endstream
endobj
```

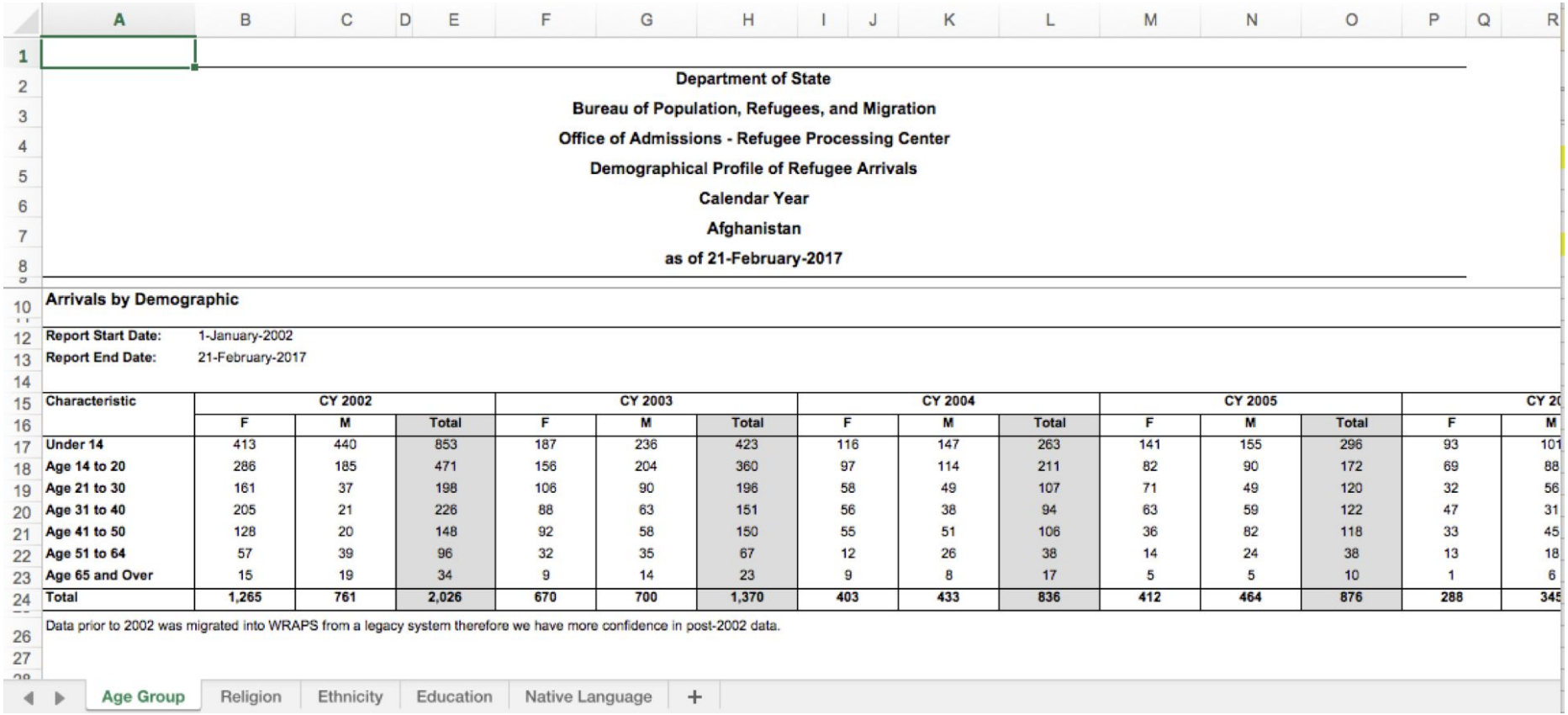
????JFIF??C

!"$"$??gg"??

???}!1AQa"q2??#B??R??$3br?
%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz???
????????

???w!1AQaq"2B????   #3R?br?
$4?%?&'()*56789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz??
??????????
????co?Ɐ??n?jc???e??,?;???Dg|???h5   X?+lr?TNXs\??   ??h??l?˥???kȶ?2??f,KɣT?V
?*?\n??6?^???ɹɪ K??????U
X?q??_lp?Y???Ic?%J??M?eɒv?)??C??/#Z?[????mx_Ē??&bT+9P掄 ??IGs??#?s?$??Lz??/?=?@??=k~???ls???T??-D??舥 ?dʭ??Z;Av9??
ZɯPT??k???T?OЦ??ў???ꝑ?al?V.??? t??x&t??+J??p?f2k)C??U?3?ӛvvd?X??????E?4G????[)Bz?}??PG?^?U??ɨ?<?Hk??A?Q?4u?2R???
KAX??;???qN??????I?B-X?T⌂h
(?[◎1E??+7+??2i?硼 ?ɸ?I1
OZ2G5IXre?q??Y[???S??n?.=)?Ɏ?r|?3?R??F9?Ж'?O???W??n?n~??ɤ?yX?????r\?F?q???ᵛK?]-"H?цZ??ca
O???gpA?mG`??a??x??`?X?^%*r?MM'cʀk?Y???y??Ө?Pm*rk?P?lL]?{yUA???b?+?3??+kK?x??L???s?ЧGWe`??"?:??Q):???3[Fź??.!r??
?fkzBIvAPA>????X?e?h\U??p?$#&??~K?Ã¿??????V:{??b
???U?.?B=h?%&??dhGtX???\~?`?;J?w?+?C?|?????m? ?{P??vrBv??_DB1????_H
?r????h??z?t???ʖ)??l?qʅlYW??g]B?H<b?Z?9+]N?
???ɥ?P?????F,?{?i?#?x??w??5???&Ui<zTdʓE?:Ⴙ??B$?????{Z?/??
?ɚ?z?g

Is this a plain text file?



Department of State

Bureau of Population, Refugees, and Migration

Office of Admissions - Refugee Processing Center

Demographical Profile of Refugee Arrivals

Calendar Year

Afghanistan

as of 21-February-2017

Arrivals by Demographic

Report Start Date: 1-January-2002

Report End Date: 21-February-2017

Characteristic	CY 2002			CY 2003			CY 2004			CY 2005			CY 20	
	F	M	Total	F	M	Total	F	M	Total	F	M	Total	F	M
Under 14	413	440	853	187	236	423	116	147	263	141	155	296	93	101
Age 14 to 20	286	185	471	156	204	360	97	114	211	82	90	172	69	88
Age 21 to 30	161	37	198	106	90	196	58	49	107	71	49	120	32	56
Age 31 to 40	205	21	226	88	63	151	56	38	94	63	59	122	47	31
Age 41 to 50	128	20	148	92	58	150	55	51	106	36	82	118	33	45
Age 51 to 64	57	39	96	32	35	67	12	26	38	14	24	38	13	18
Age 65 and Over	15	19	34	9	14	23	9	8	17	5	5	10	1	6
Total	**1,265**	**761**	**2,026**	**670**	**700**	**1,370**	**403**	**433**	**836**	**412**	**464**	**876**	**288**	**345**

Data prior to 2002 was migrated into WRAPS from a legacy system therefore we have more confidence in post-2002 data.

Age Group | Religion | Ethnicity | Education | Native Language | +

,,,
Department of State,,,
"Bureau of Population, Refugees, and Migration",,,
Office of Admissions - Refugee Processing Center,,,
Demographical Profile of Refugee Arrivals,,,
Calendar Year,,,
Afghanistan,,,
as of 21-February-2017,,,

,,,
Arrivals by Demographic,,,

,,,
Report Start Date:,1-January-2002,,
Report End Date:,21-February-2017,,

,,,
Characteristic,CY 2002,,,CY 2003,,CY 2004,,,CY 2005,,CY 2006,,,CY 2007,,CY 2008,,CY 2009,,CY 2010,,CY 2011,,CY 2012,,CY 2013,,CY 2014,,CY 2015,,CY 2016,,CY 2017,,Cumulative Total,%
,F ,M ,Total,,F ,M ,Total,F ,,M ,Total,F ,M ,Total,F ,,M ,Total,F ,M ,Total,F ,M ,Total,F ,M ,Total,F ,M ,Total,F ,M ,Total,F ,M ,Total,F ,M ,Total,F ,M ,Total,F ,M ,Total,F ,M ,Total,F ,M ,Total,,Total
Under 14,413,440,853,,187,236,423,116,,147,263,141,155,296,93,,101,194,65,55,120,78,85,163,55,51,106,75,78,153,44,61,105,83,96,179,84,106,190,102,100,202,145,190,335,464,447,911,54,76,130,"4,623",31.96%
Age 14 to 20,286,185,471,,156,204,360,97,,114,211,82,90,172,69,,88,157,34,55,89,63,66,129,29,43,72,48,68,116,47,38,85,56,79,135,63,92,155,97,87,184,120,197,317,311,385,696,36,58,94,"3,443",23.80%
Age 21 to 30,161,37,198,,106,90,196,58,,49,107,71,49,120,32,,56,88,39,24,63,50,42,92,24,32,56,45,67,112,46,37,83,41,48,89,66,64,130,82,52,134,119,91,210,303,277,580,39,25,64,"2,322",16.05%
Age 31 to 40,205,21,226,,88,63,151,56,,38,94,63,59,122,47,,31,78,34,26,60,47,42,89,31,26,57,35,38,73,30,32,62,50,43,93,57,37,94,51,27,78,82,61,143,225,138,363,29,23,52,"1,835",12.69%
Age 41 to 50,128,20,148,,92,58,150,55,,51,106,36,82,118,33,,45,78,20,23,43,38,53,91,15,22,37,24,17,41,23,12,35,31,21,52,35,2

Plain Text Editors

Integrated Development Environments (IDEs) for coding in Python let you write plain text files.

Popular Python IDEs: Jupyter Lab, PyCharm, VSCode, Spyder

You can also choose a standalone text editor:

Built-in options: Mac: TextEdit PC: Notepad

Other options: Nano, Vim, Atom, Emacs, Sublime, BBEdit, TextWrangler, many more

Time to Review!

Data Types

Numbers

Integers

```
-38291423
0
2
```

Floats

```
3.0
-432.2343253
4.938e-10
```

String

AKA: text, characters

Enclosed in single or double quotation marks.

```
"This is a string"
```

```
'This is a string 2'
```

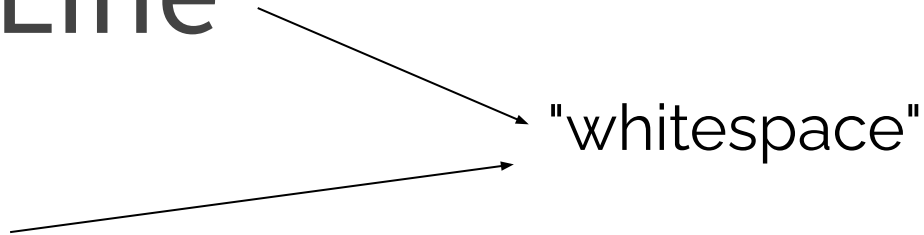
"" (**empty** string)

" " (this is NOT an empty string)

Special Characters

\n New Line

\t Tab



"This is line 1.\nThis is line 2."

Case and type matter

"A" is not equal to "a"

"3" (string) is not the same as integer 3

Sorting Strings: Alphabetical Order

```
"110 cats"

"3 cats"

"Apple"

"Mushroom"

"apple"

"mushroom"
```

String Indexing (aka string slicing)

"String indexing is fun"

	S	t	r	i	n	g		i	n	d	e	x	i	n	g		i	s
Python	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
R	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

String Indexing (aka string slicing)

"String indexing is fun"

	S	t	r	i	n	g		i	n	d	e	x	i	n	g		i	s
Python	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
R	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

"Apple"[4] returns "e"

"Apple"[-1] returns "e"

String Indexing (aka string slicing)

"String indexing is fun"

	S	t	r	i	n	g		i	n	d	e	x	i	n	g		i	s
Python	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

produces substrings

"Apple"[0:3] returns "App"

Joining Strings: Concatenate

```
"Red" + "bull" = "Redbull"

"Red" + " " + "bull" = "Red bull"
```

Boolean

TRUE

FALSE

TRUE FALSE T F True False

Boolean Operators

NOT: `!` `not`

AND: `&` `and`

OR: `|` `or`

Boolean Operators: AND

```
TRUE and TRUE = TRUE

TRUE and FALSE = FALSE

FALSE and TRUE = FALSE

FALSE and FALSE = FALSE
```

Boolean Operators: OR

```
TRUE or TRUE = TRUE

TRUE or FALSE = TRUE

FALSE or TRUE = TRUE

FALSE or FALSE = FALSE
```

Boolean Operators: NOT

```
not TRUE = FALSE

not FALSE = TRUE

TRUE and not TRUE = FALSE

TRUE and not FALSE = TRUE
```

Boolean Operators: Grouping

```
(TRUE and FALSE) or
  not (FALSE and TRUE) =
```

Boolean Operators: Grouping

```
FALSE or
   not FALSE =
```

Boolean Operators: Grouping

```
FALSE or
   TRUE =
```

Boolean Operators: Grouping

```
FALSE or
   TRUE = TRUE
```

Converting Between Data Types

```
TRUE as integer = 1

FALSE as integer = 0

3.5 as string = "3.5"
```

Special Types

NULL

None

Missing Data: NA

Time to Review!

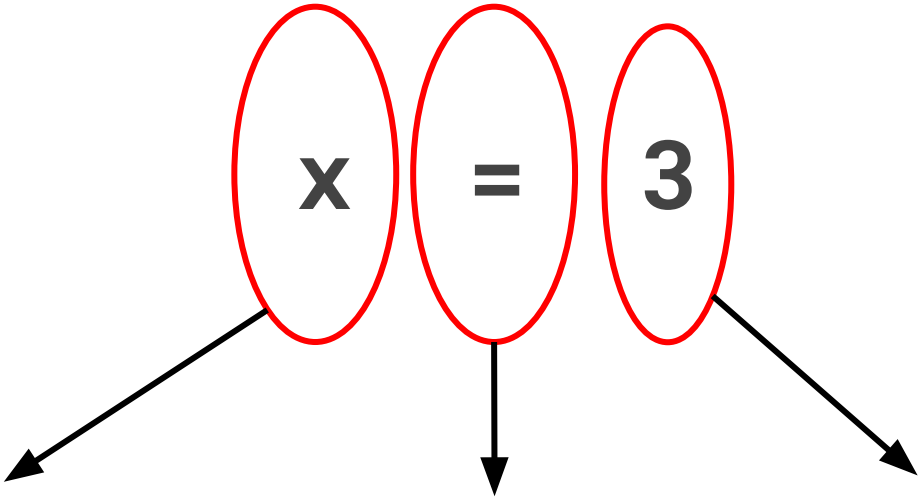
Variables

Variables

Variables let us refer to a value with a name. We can use the same name, but change the value.

Variables can be used to name integers, floats, strings, lists, arrays, equations, dictionaries, dataframes, the text in files, and more.

Variables



Variable name

"Left hand side"

Assignment operator

Value

"Right hand side"

Variables

```
x = 3 + 5

x
```

```
x is 8
```

Variables

```
x = 3

x + 5

  x
```

x is 3

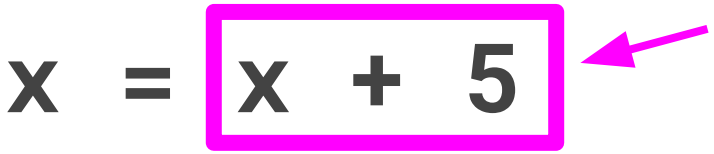
Variables

```
x = 3

x = x + 5

x
```

x is 8

Variables

```
x = 3

x = x + 5

x
```



Everything on the right side is evaluated first - before the variable is assigned

```
x is 8
```

Variables

```
x = 3
y = 5
x = x + y
x
```

x is 8

Variables

```
cats = 3
kittens = 5
cats = cats + kittens
kittens = 7
cats
```

cats is 8

Variable Names

Case matters

Start with a letter

Make names meaningful

Style conventions

- camelCase
- separate_with_underscores

Time to Review!

Lists, vectors, arrays

Multiple Related Values

```
Ages of students: 42, 30, 24, 24, 27, 35, 39, 22
```

Lists, Vectors, Arrays

Stores the items in the order given.

Ages of students: 42, 30, 24, 24, 27, 35, 39, 22

Lists, Vectors, Arrays

Python list - enclosed in square brackets

```
ages = [42, 30, 24, 24, 27, 35, 39, 22]
```

R vector - created with the c() function

```
ages <- c(42, 30, 24, 24, 27, 35, 39, 22)
```

Lists, Vectors, Arrays

```
students = ["Michael", "Chen", "Yishu", "June", "Amy"]
```

```
evanston_resident = [True, False, True, False, True]
```

```
sample_vals = [3.544, 10.0, 18.32]
```

Nested Lists

```
[ [1, 2, 3], ["a", "b", "c", "d"] ]
```

Lists, Vectors, Arrays

```
Length = number of elements

len([42, 30, 24, 24, 27, 35, 39, 22]) = 8


Empty list

[]
```

List Indexing (Python)

```
ages = [42, 30, 24, 24, 27, 35, 39, 22]
         0   1   2   3   4   5   6   7
ages[1]
```

30

List Indexing (Python)

```
ages = [42, 30, 24, 24, 27, 35, 39, 22]
          0   1   2   3   4   5   6   7
ages[1:4]
```

[30, 24, 24]

Python indexing is

exclusive of the end position

List Variables

```
ages = [42, 30, 24, 24, 27, 35, 39, 22]
ages[1] = 54
ages
```

[42, 54, 24, 24, 27, 35, 39, 22]

List Variables

```
ages = [42, 30, 24, 24, 27, 35, 39, 22]
ages = 54
ages
```

54

Appending and Prepending

[42, 30, 24, 24, 27, 35, 39, 22]

Append: [42, 30, 24, 24, 27, 35, 39, 22, **50**]

There is no prepending in Python.

Time to Review!

Conditions

Conditions

Expressions that produce a boolean value:

True or False

Comparisons

`>` `<` `>=` `<=`

`==` `!=`

Comparisons

```
age = 13
```

Assignment

```
age < 21
```

```
age == 14
```

Comparisons

Comparisons

```
age = 13
```

```
age < 21
```

```
age == 14
```

Comparisons

```
age = 13
```

```
age < 21
```

TRUE

```
age == 14
```

Comparisons

```
age = 13
```

```
age < 21
```

TRUE

```
age == 14
```

FALSE

Compound Conditions

```
first_age = 13

second_age = 15
```

```
first_age < 21 or second_age < 21
```

Compound Conditions

```
first_age = 13

second_age = 15
```

```
first_age < 21 or second_age < 21

TRUE or TRUE = TRUE
```

Compound Conditions

```
first_age = 13
```

```
first_age <= 19 and first_age > 13
```

Compound Conditions

```
first_age = 13
```

```
first_age <= 19 and first_age > 13
```

TRUE and FALSE = FALSE

Compound Conditions

```
first_age = 13
```

```
first_age <= 19 and first_age > 13
```

Notice that you have to include "first_age" on both sides of the "and"

Conditions with Booleans

```
evanston_resident = True

nu_staff = True
```

```
evanston_resident == True and nu_staff == True
```

Conditions with Booleans

```
evanston_resident = True

nu_staff = True
```

~~`evanston_resident == True and nu_staff == True`~~

`evanston_resident and nu_staff`

`evanston_resident & nu_staff`

Conditions with Booleans

```
evanston_resident = True

nu_staff = True
```

not evanston_resident and nu_staff

```
!evanston_resident & nu_staff
```

Conditions with Booleans

```
evanston_resident = False

nu_staff = False
```

`not evanston_resident and not nu_staff`

```
!evanston_resident & !nu_staff
```

Control Flow: if/then/else

If Statements

```
if condition:
    do something
```

If Statements

```
if condition:
    do something
```

Evaluates to a single TRUE or FALSE value

If Statements

```
if age >= 18:
    print("Go Vote!")
```

If Statements

```
if condition:
    do something
else:
    do something different
```

If Statements

```
if age >= 18:
  print("Go Vote!")
else:
  print("Too young!")
```

Chained If Statements

```
if condition:
    do something
else if condition2:
    do something different
else:
    do a third thing
```

Chained If Statements

```
if age >= 18:
   print("Go Vote!")
elif age >= 16:
   print("Learn to drive")
else:
   print("Too young!")
```

Chained If Statements

```
if age >= 18:
   print("Go Vote!")
elif age >= 16:
   print("Learn to drive")
else:
   print("Too young!")
```

18, 19, 20, 21...

16, 17

15, 14, 13, 12, 11, 10...

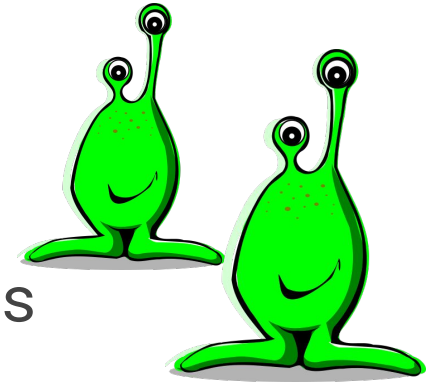
If Statements: Multiple Actions

```
if age >= 18:
    print("Go Vote!")
elif age >= 16:
    print("Learn to drive")
    print("Stay in school")
else:
    print("Too young!")
```

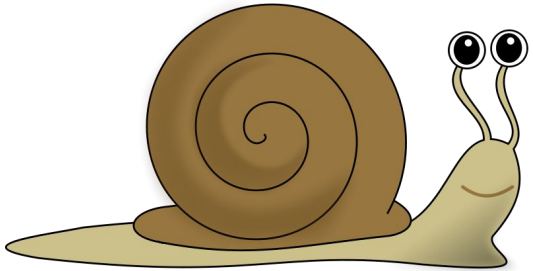
If Statements: includes indentation

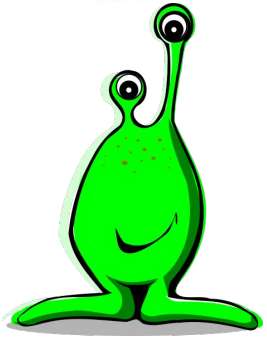
```
if age >= 18:
    print("Go Vote!")
elif age >= 16:
    print("Learn to drive")
    print("Stay in school")
else:
    print("Too young!")
```

Writing Pseudocode



```
if color is green and has eye stalks
    send to Mars
else
    send to Earth
```

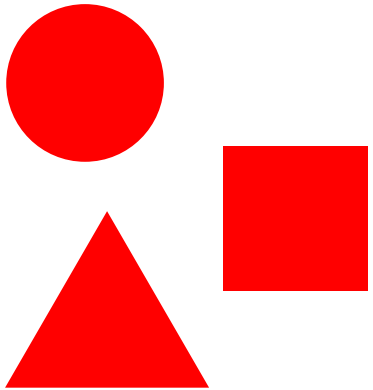




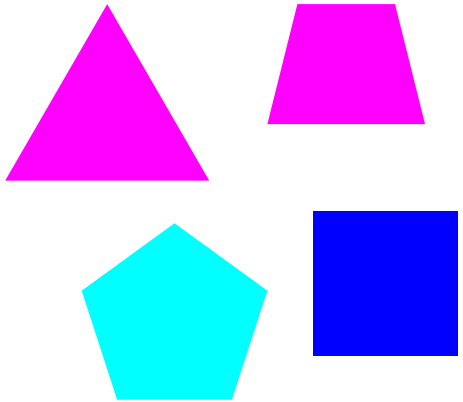
Exercise

Write an if/else statement that would sort the shapes below into the correct groups

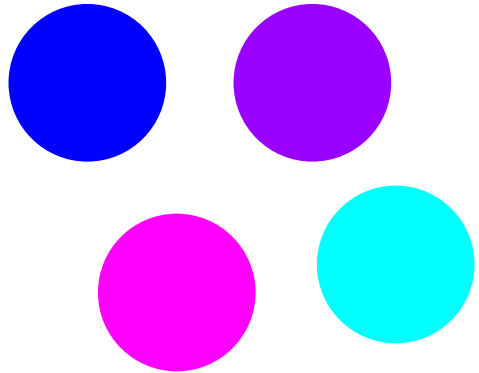
Group 1



Group 2



Group 3



Exercise

```
if shape is red:
  Put in group 1
else if shape is circle:
  Put in group 3
else:
  Put in group 2
```

If Statements and Lists/Vectors

```
ages = [42, 30, 24, 24, 27, 35, 39, 22]
```

```
if ages > 18
   Do something
```

Time to Review!

Loops

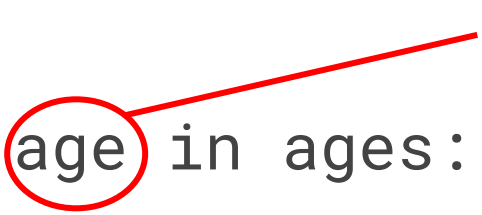
For Loop

Loops are how we repeat the same action many times

```
for variable in list of values:
    do something to or with the variable
```

For Loop

```
ages = [42, 30, 24, 24, 27, 35, 39, 22]
```




```
for age in ages:
    print(age)
```

```
for i in ages:
    print(i)
```

```
for dog in row

    bark(dog)
```

Ruff!

```
for dog in row

bark(dog)
```

```
for dog in row

    bark(dog)
```

```
for dog in row

    bark(dog)
```

```
for dog in row

    bark(dog)
```

```
for dog in row

    bark(dog)
```

```
for dog in row

bark(dog)
```

```
for dog in row

    bark(dog)
```

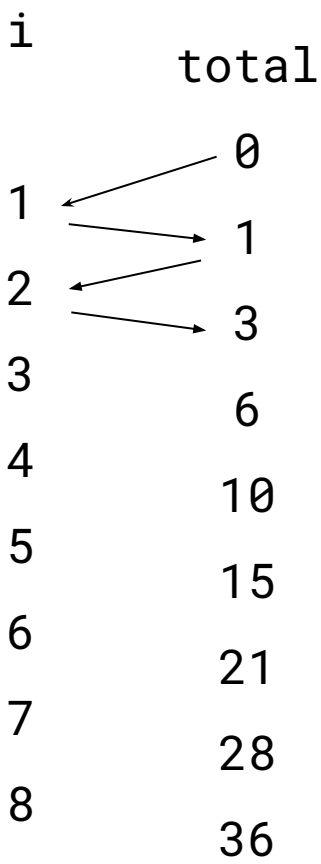
For Loop

```
total = 0

for i in range(1:8):

    total = total + i

print(total)
```

1, 2, 3, 4, 5, 6, 7, 8

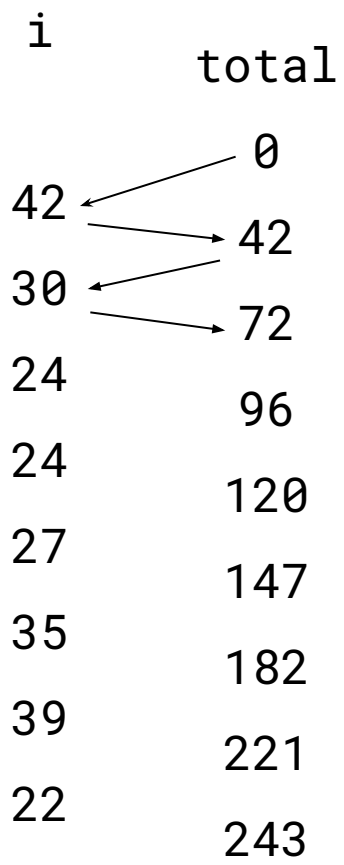
For Loop

```
total = 0

for i in range(1:8):

    total = total + i

print(total)
```

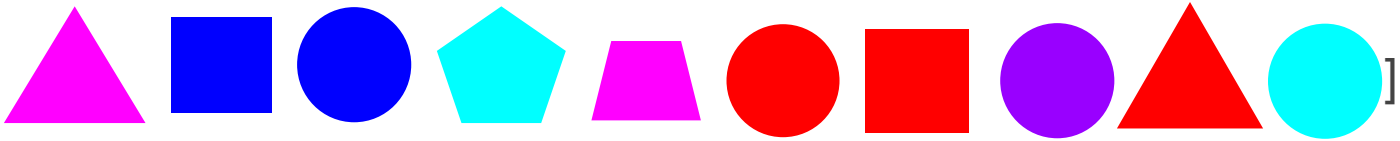


For Loop

```
total = 0

ages = [42,30,24,24,27,35,39,22]


for i in ages:

   total = total + i

print(total)
```



For Loop with if/else statements

shapes = []

```
for shape in shapes:
    if shape is red:
        Put in group 1
    else if shape is circle:
        Put in group 3
    else:
        Put in group 2
```

pseudocode!

```
for dog in row:
    if dog ears up:
        bark(dog)
```


```
for dog in row
    if dog ears up
        bark(dog)
```

```
for dog in row
    if dog ears up
        bark(dog)
```

```
for dog in row
    if dog ears up
        bark(dog)
```

```
for dog in row
    if dog ears up
        bark(dog)
```

Exercise for practicing loops at home:

https://www.google.com/logos/2017/logo17/logo17.html

Or search Google for "google doodle coding".

Time to Review!

Functions

Functions

- Functions are commands that will do something to or with your data (integers, floats, strings, lists, etc.)

Functions

- Take input values called **arguments**
- Can **return** an output value
- Python and functions don't alter their input values when working with numbers and strings (some functions alter lists - we'll talk about this in the boot camp.)

Calling Functions: Return Values

Some functions return a value:

```
abs(-3)
```

Returns 3

Calling Functions: Return Values

Some functions return a value:

```
abs(-3)
```

Returns 3

-3 is an **argument** that we **passed** to the function abs()

Calling Functions: Return Values

Some functions return a value:

```
abs(-3)
```

Returns 3

The return value can be assigned to a variable:

```
x = abs(-3)
```

x is 3

Calling Functions: No Return Value

Other functions do not return a value, but are called to DO something:

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

```
Hello World!
```

They cannot be assigned to a variable:

```
x = print("Hello World!")
```

Calling Functions: Variable Arguments

Arguments can be variables:

```
x = "Hello World!"

print(x)


file = "results.csv"

open(file)
```

Function Definitions

The open() function in Python only requires 1 argument - the file name

```
file = "results.csv"

open(file)
```

However, open() can take other optional arguments that will change the way the function works

Function Definitions

You can find lists of all the possible arguments that can be passed to a function in the function definition, which is found in the **documentation** for the language.

https://docs.python.org/3/

Function Definitions

`open`(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

Function Definitions: Function Name

`open`(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

Function Definitions: Parameters

`open`(*file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None*)

Function Definitions: Parameter Names

`open`(*file*, *mode*='r', *buffering*=-1, *encoding*=None, *errors*=None, *newline*=None, *closefd*=True, *opener*=None)

Function Definitions: Default Values

`open`(*file*, *mode*=*'r'*, *buffering*=*-1*, *encoding*=*None*, *errors*=*None*, *newline*=*None*, *closefd*=*True*, *opener*=*None*)

Function Definitions: Required Parameters

`open(`***file***`, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)`

Function Definitions: Default Values

`open`(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

If a parameter has a default value, you do not have to pass an argument for that parameter. It will just use the default.

Calling Functions

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

```
open("results.txt", mode='w')
```

Calling Functions: Arguments

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

```
open("results.txt", mode='w')
```

Calling Functions: Keyword Arguments

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

```
open("results.txt", mode='w')
```

Calling Functions: Non-Keyword Arguments

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

```
open("results.txt", mode='w')

open("results.txt", 'w')
```

Calling Functions: Non-Keyword Arguments

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

```
open("results.txt", 'w')
```

```
open("w", 'results.txt')
```

You can pass a keyword argument without the keyword if the arguments are in the same order as they are in the function definition. **Order matters.**

Calling Functions: Argument order

open(*file*, *mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*, *closefd=True*, *opener=None*)

```
open(file="results.txt", mode='w')
```

If you are unsure of the order, you can always use the parameter name with all your arguments.

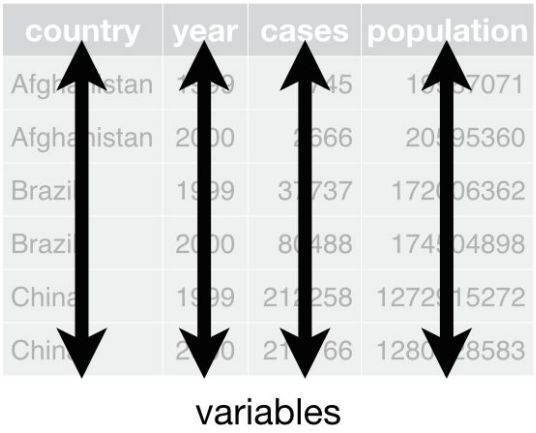
Packages/Libraries/Modules

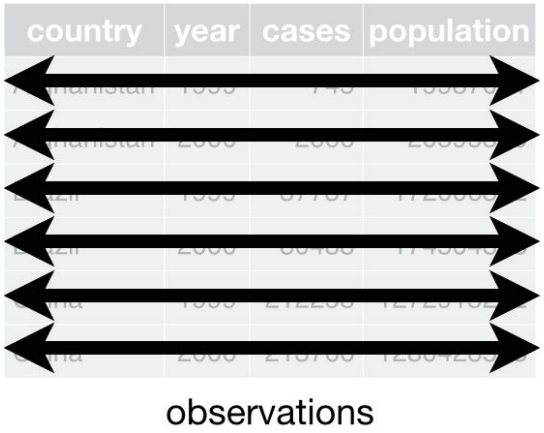
- Collections of functions, data, and other code
- Some must be installed, others are standard
- **installed** - it has been downloaded to the computer
- **imported** - it has been loaded into the current script or interactive instance (must happen before you can use it)
- Using packages/libraries/modules is expected!
- Look for pre-existing code/solutions
    - How do you know it's good/correct?

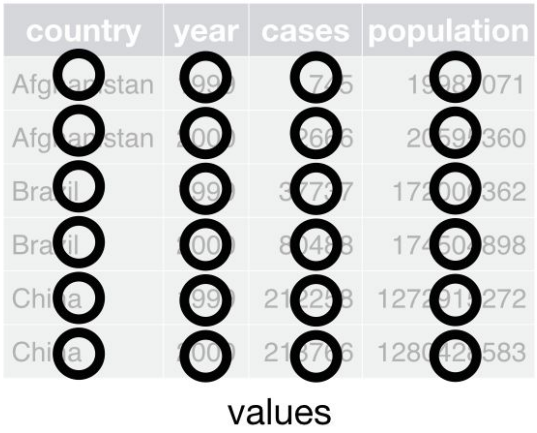
Time to Review!

Data

Rectangles of Data







Rows: Observations

Person (or mouse, worm, etc.)

Country

Year

Run/trial of an experiment

Chemical

Sample

Columns: Variables

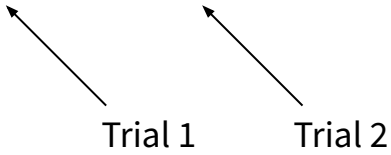
Measurements

Grouping/identification variables:
- > trial/sample
- > condition
- > label for the observation (country name)

Each ID variable in its own column

Exercise - reshape this data

	1 min				5 min			
strain	normal		mutant		normal		mutant	
A	111	170	375	384	277	234	207	466
B	336	169	491	233	392	341	213	472



strain	genotype	minute	trial	response
A	normal	1	1	111
A	normal	1	2	170
A	mutant	1	1	375
A	mutant	1	2	384
A	normal	5	1	277
A	normal	5	2	234
A	mutant	5	1	207
A	mutant	5	2	466
B	normal	1	1	336
B	normal	1	2	169
B	mutant	1	1	491
B	mutant	1	2	233
B	normal	5	1	392
B	normal	5	2	341
B	mutant	5	1	213
B	mutant	5	2	472

QUESTIONS?