Programming Concepts

with instructor Colby Witherup Wood

will begin at **1:02 pm Central**

This workshop is brought to you by

NUIT Research Computing 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

Downloading materials from GitHub

https://github.com/agithasnoname/programmingConcepts

Click on the green **Clone or download** button

then click on **Download ZIP**

During this workshop, ask questions in the chat. I will answer them periodically throughout the workshop.

If my internet goes out during this workshop, that means you get a 10 minute break! You may have to log back into Zoom. Sorry!
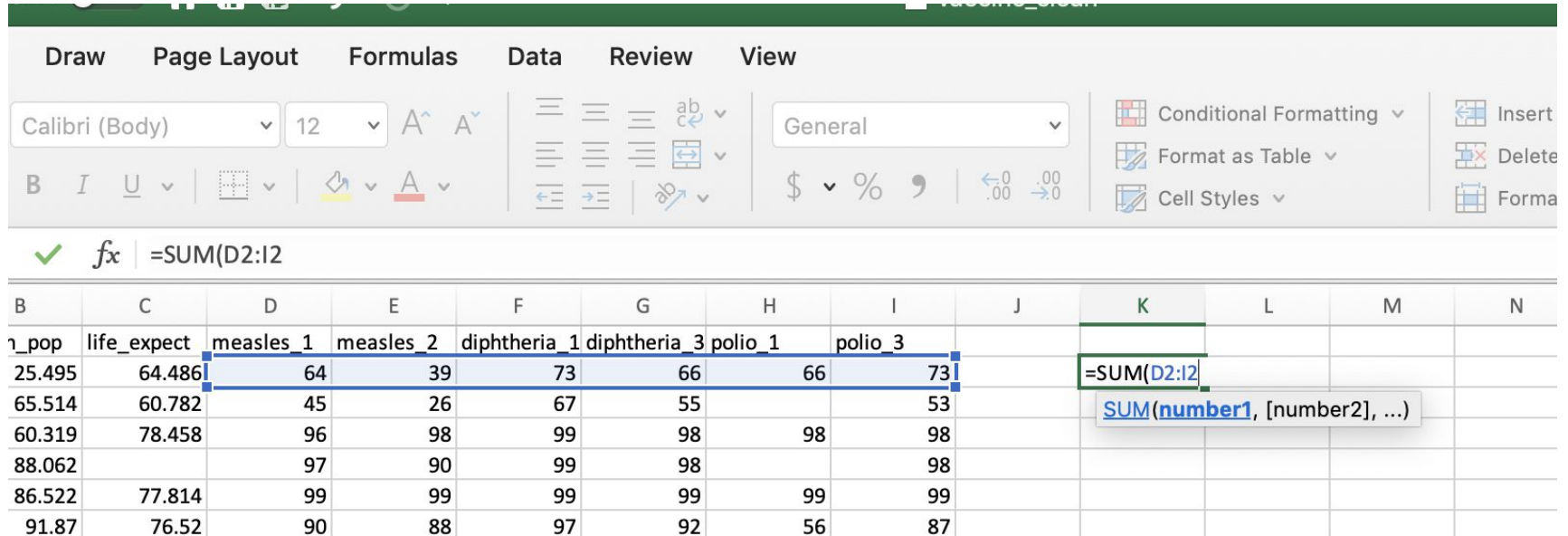
Programming languages

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

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.
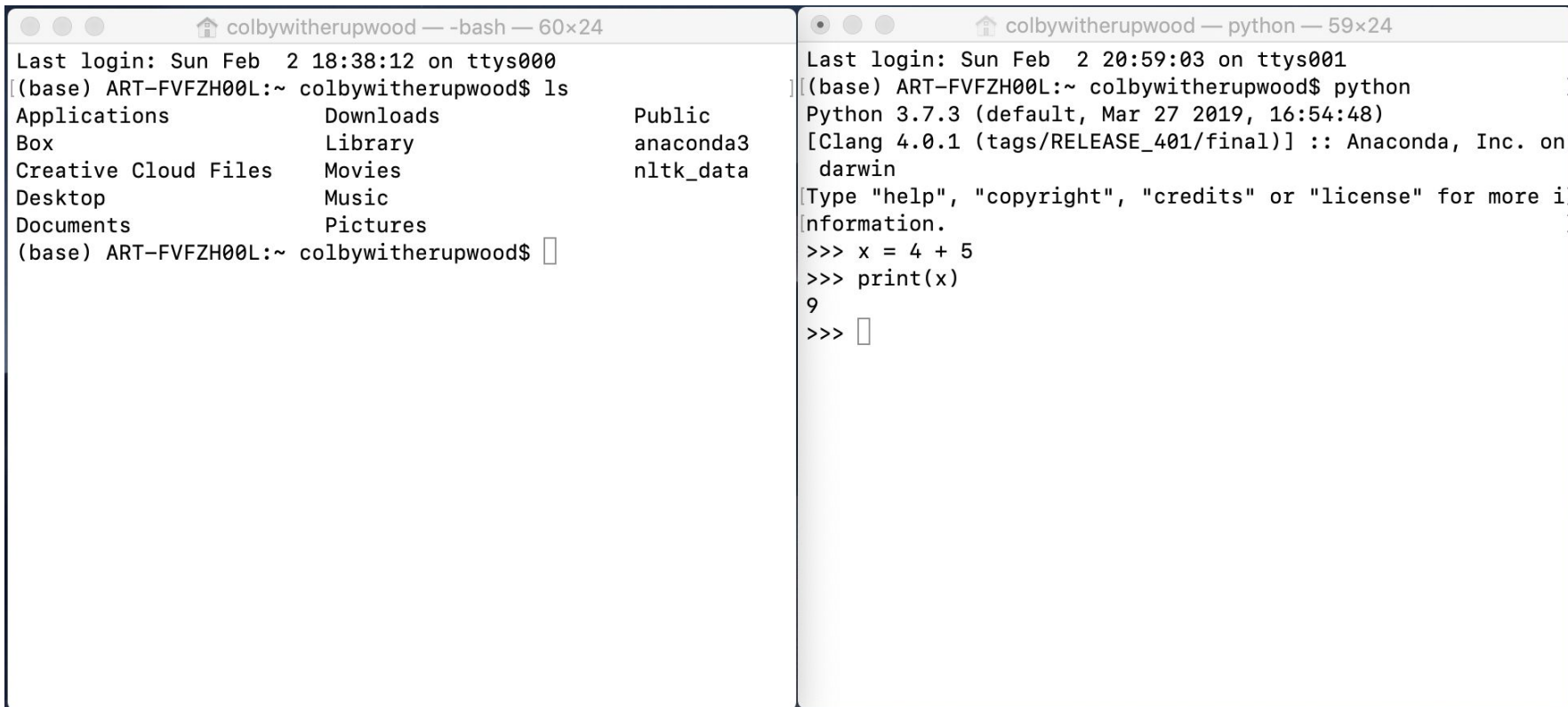
Mac: Terminal uses Unix Bash, PC: Windows PowerShell

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)

- With the help of a GUI. GUIs for coding are called **IDEs - Integrated Development Environments**

Command prompts


```
Last login: Sun Feb  2 18:38:12 on ttys000
[(base) ART-FVFZH00L:~ colbywitherupwood$ ls
Applications            Downloads               Public
Box                     Library                 anaconda3
Creative Cloud Files    Movies                  nltk_data
Desktop                 Music
Documents               Pictures
(base) ART-FVFZH00L:~ colbywitherupwood$ 
```

```
Last login: Sun Feb  2 20:59:03 on ttys001
[(base) ART-FVFZH00L:~ colbywitherupwood$ python
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on
 darwin
[Type "help", "copyright", "credits" or "license" for more i]
[nformation.                                                ]
>>> x = 4 + 5
>>> print(x)
9
>>> 
```

Time to Review!

Open the ProgrammingConceptsReview document from the folder you downloaded.

Filesystems

Reminder: If my internet goes out during this workshop, that means you get a 10 minute break! You may have to log back into Zoom. Sorry!

Filesystems

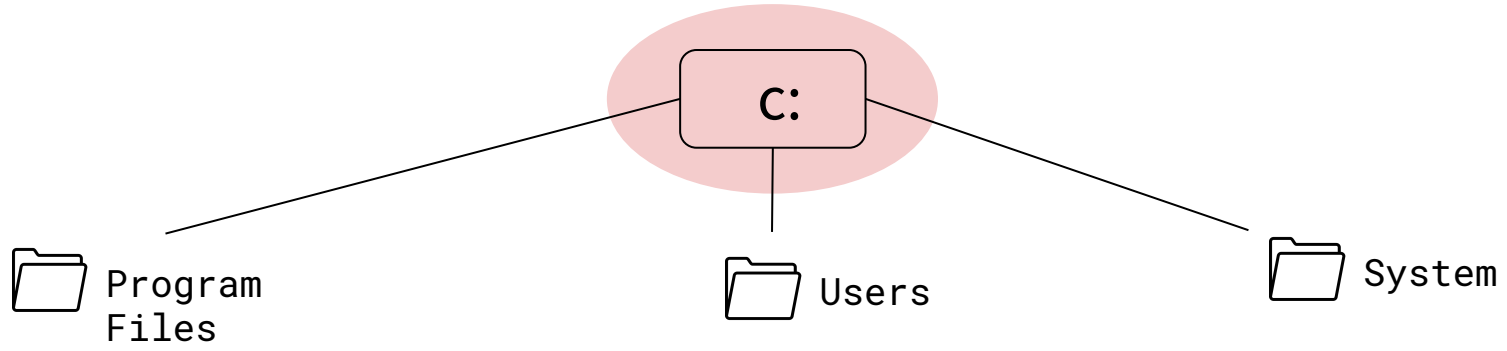Coding requires us to move away from point and click.

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**.

c:

C:\

C:\Users\
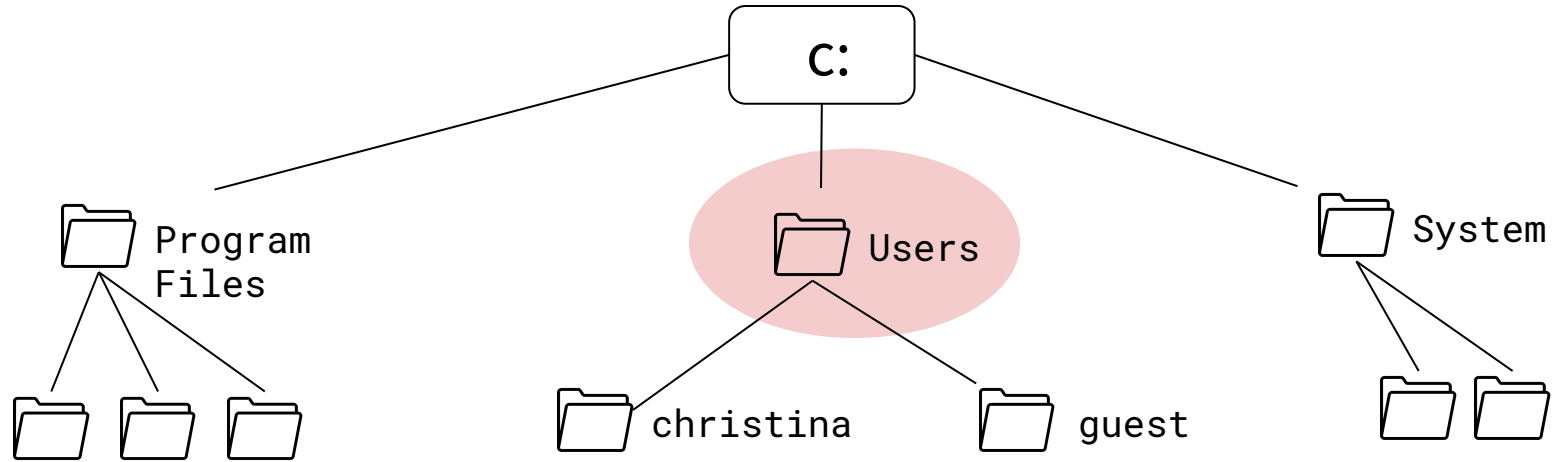
`C:\Users\christina`
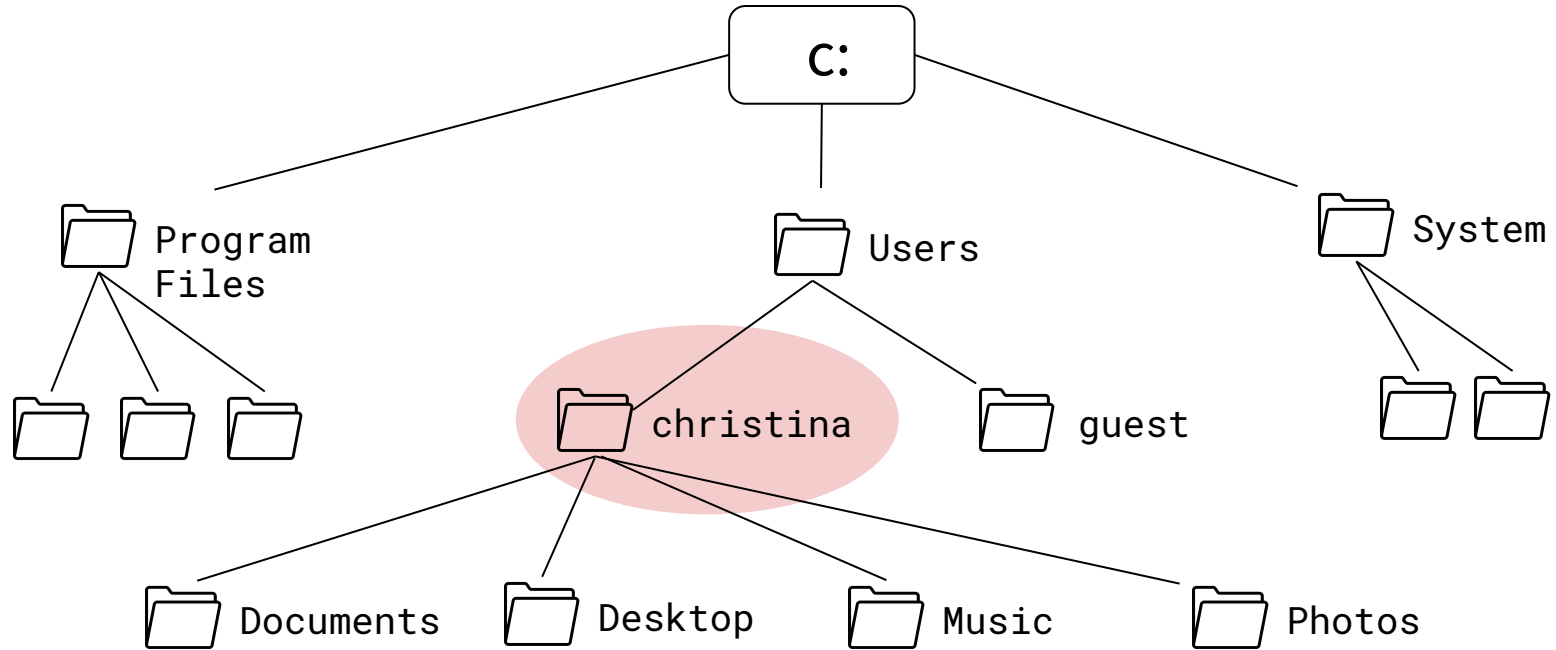
C:\Users\christina\Documents
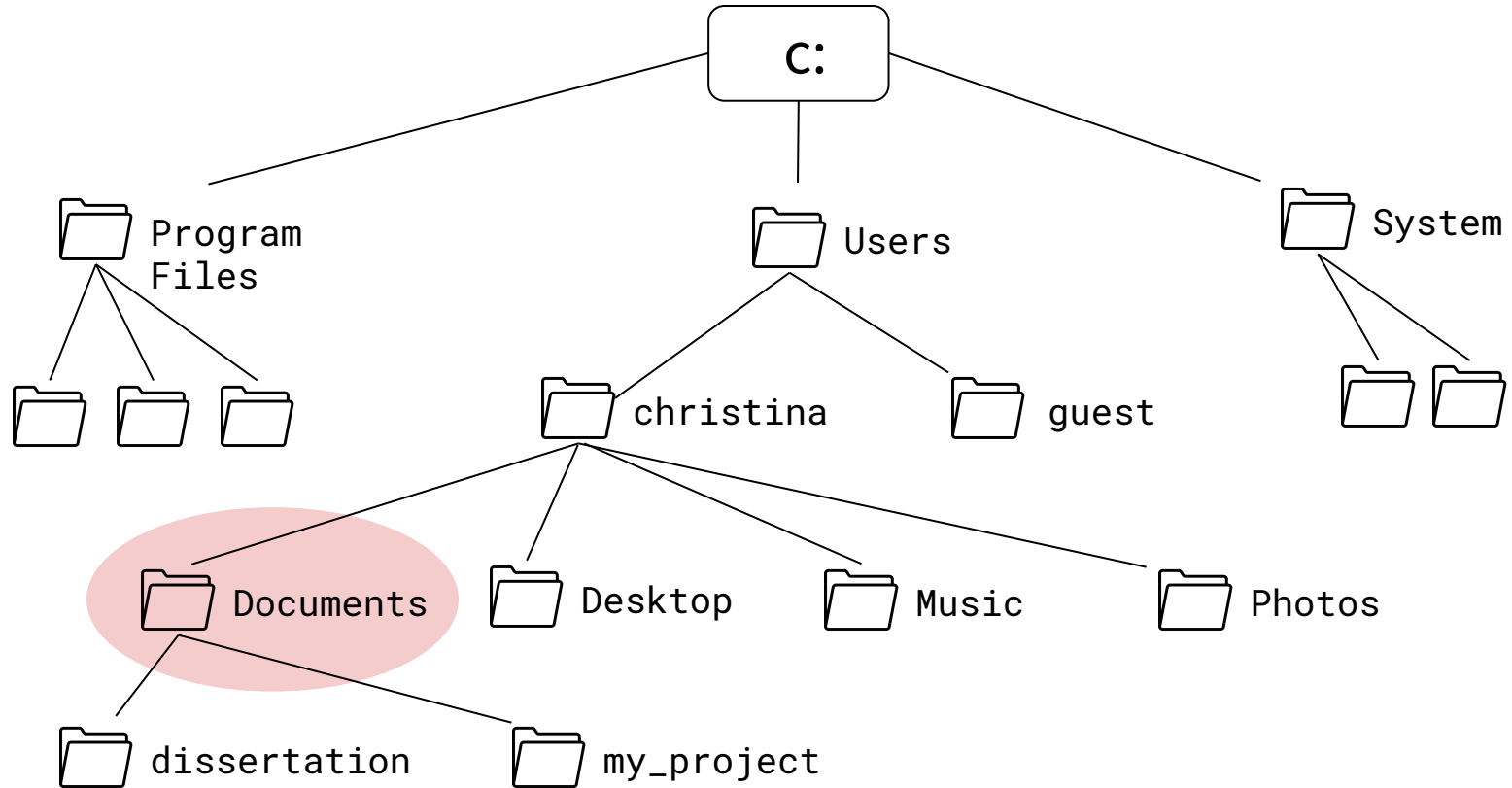
C:\Users\christina\Documents\my_project
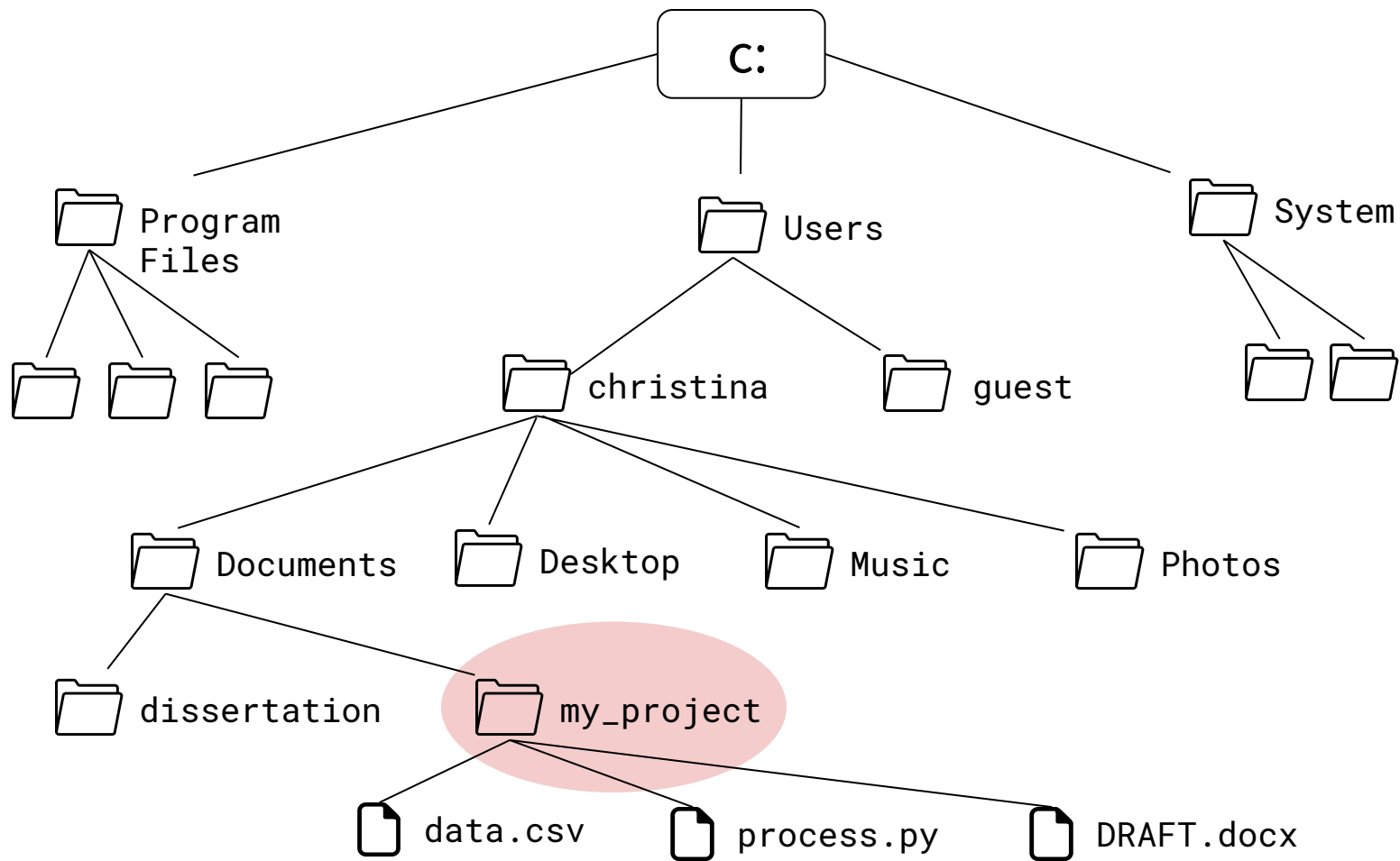
C:\Users\christina\Documents\my_project\data.csv
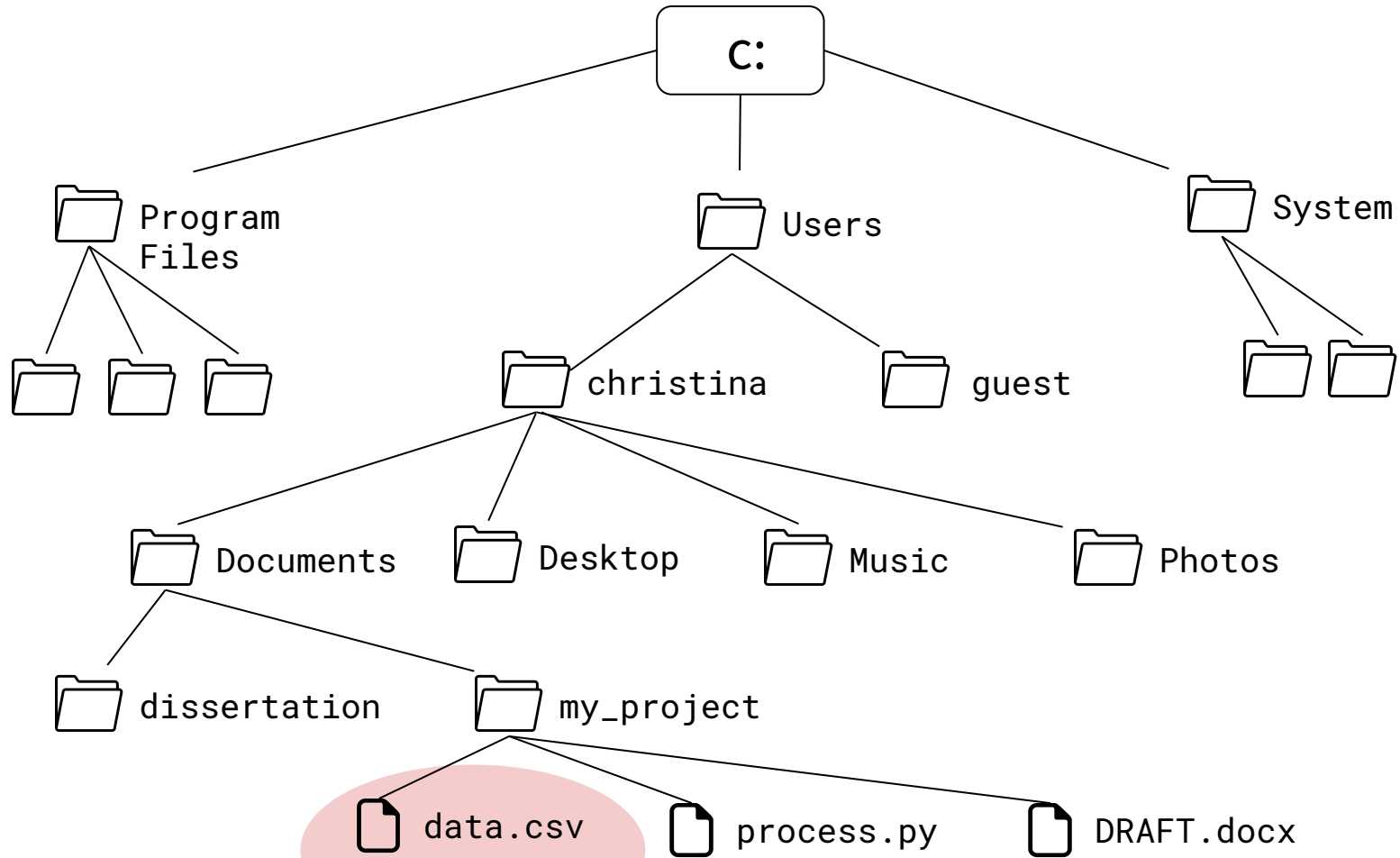
`/Users/christina/Documents/my_project/data.csv`
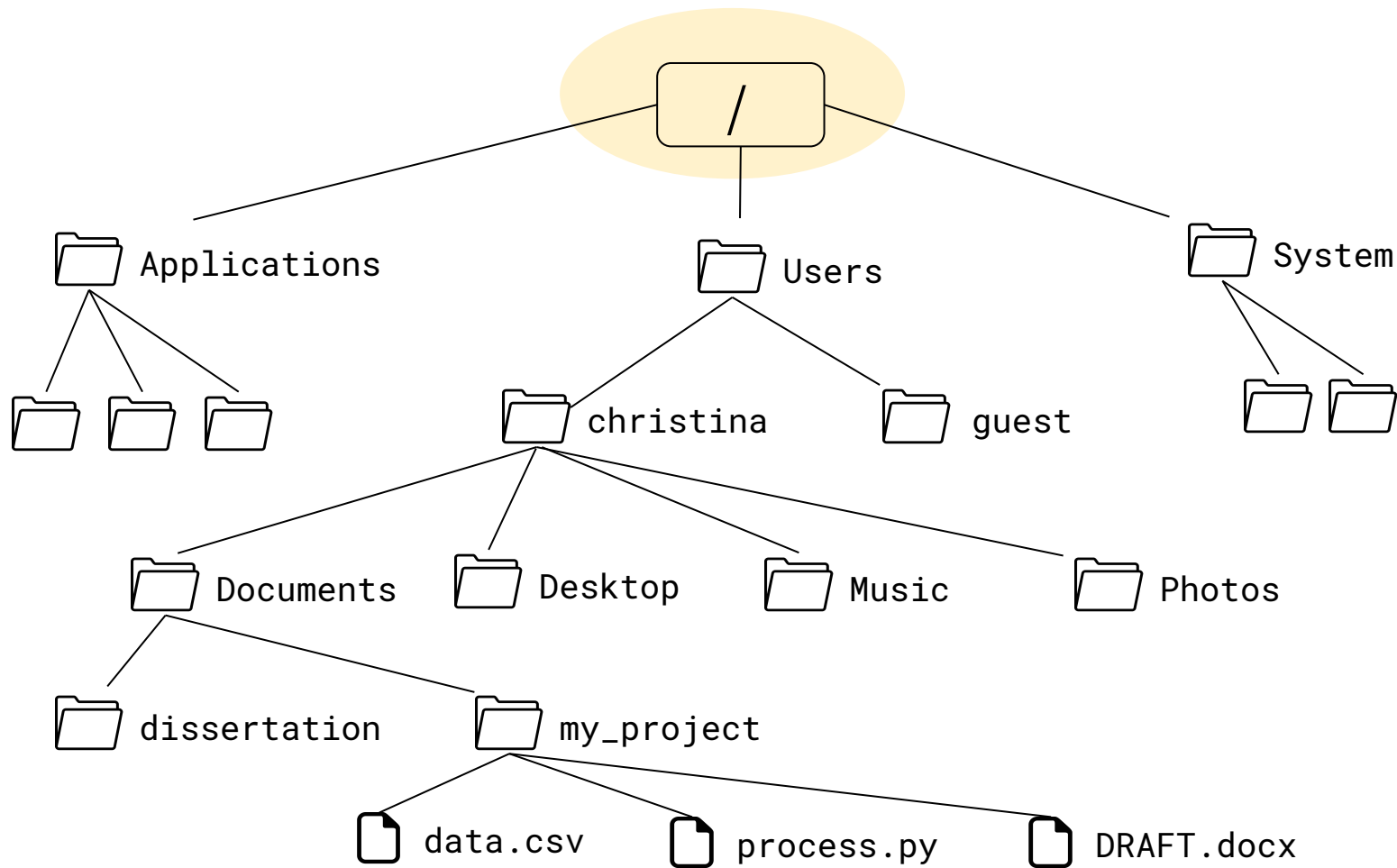
Absolute Paths

`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   =   ~
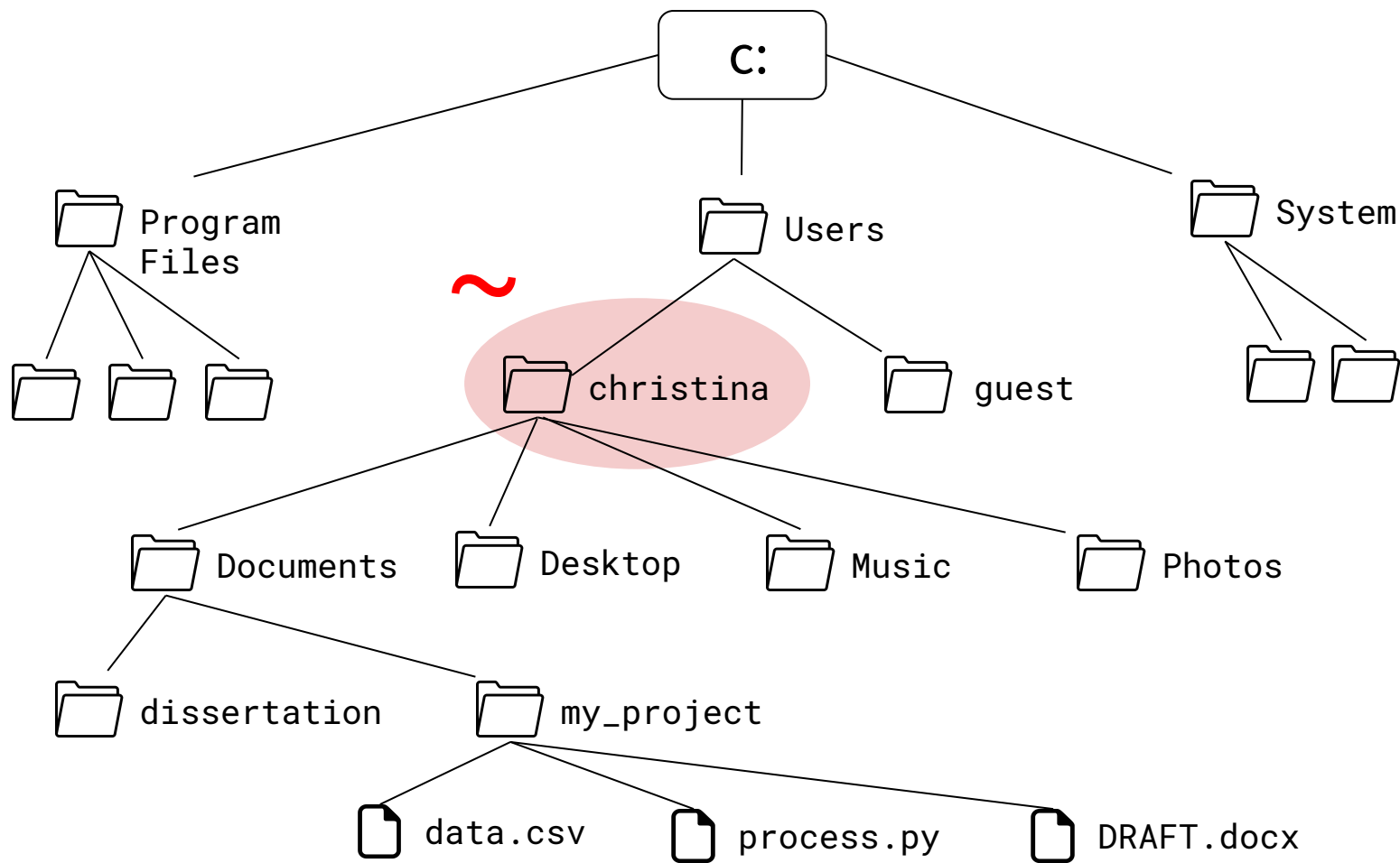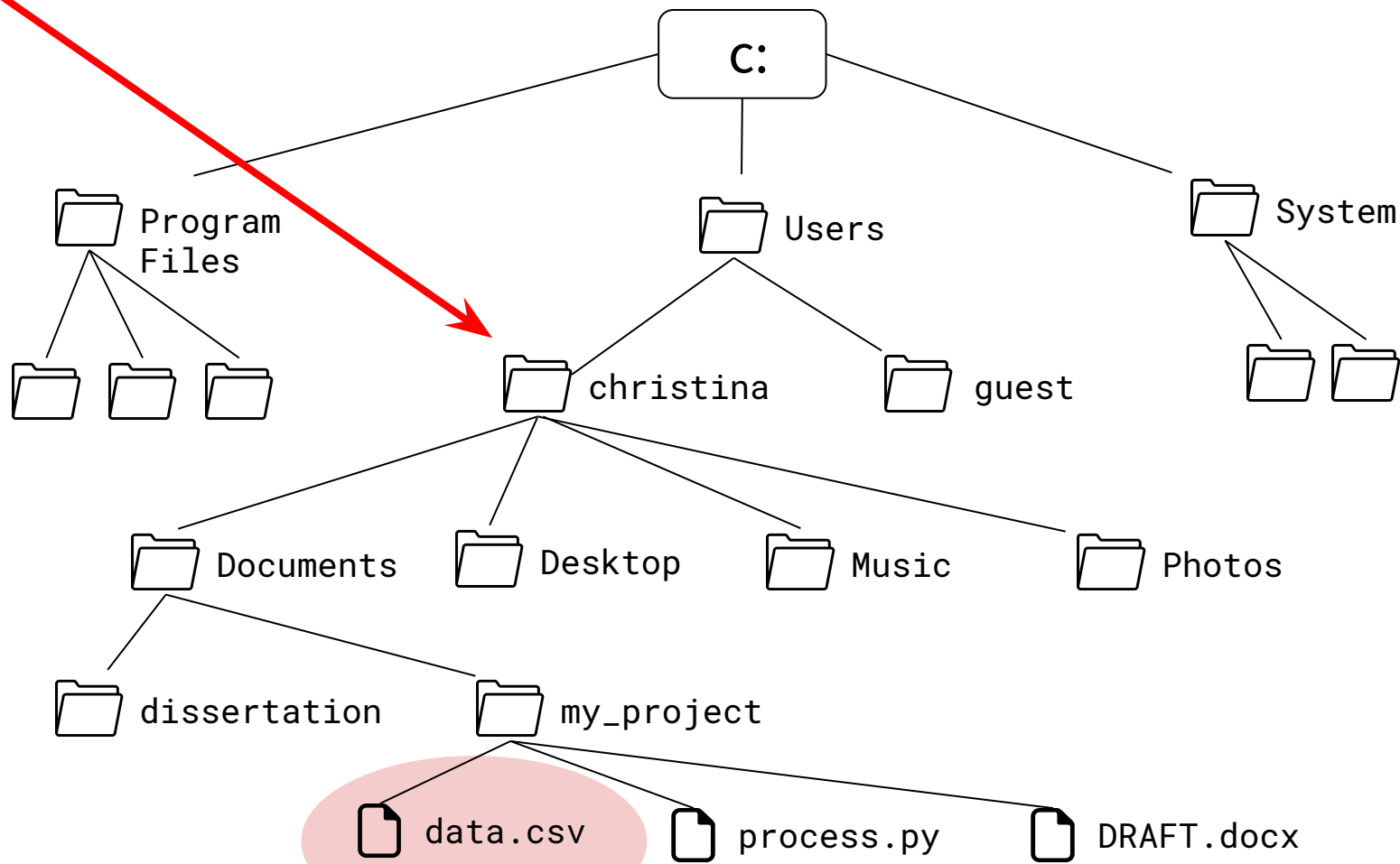
Working Directory

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:

C:
├── Program Files
│   ├── (folder)
│   ├── (folder)
│   └── (folder)
├── Users
│   ├── christina
│   │   ├── Documents
│   │   │   ├── dissertation
│   │   │   └── my_project
│   │   │   ├── data.csv
│   │   │   ├── process.py
│   │   │   └── DRAFT.docx
│   │   ├── Desktop
│   │   ├── Music
│   │   └── Photos
│   └── guest
└── System
    ├── (folder)
    └── (folder)
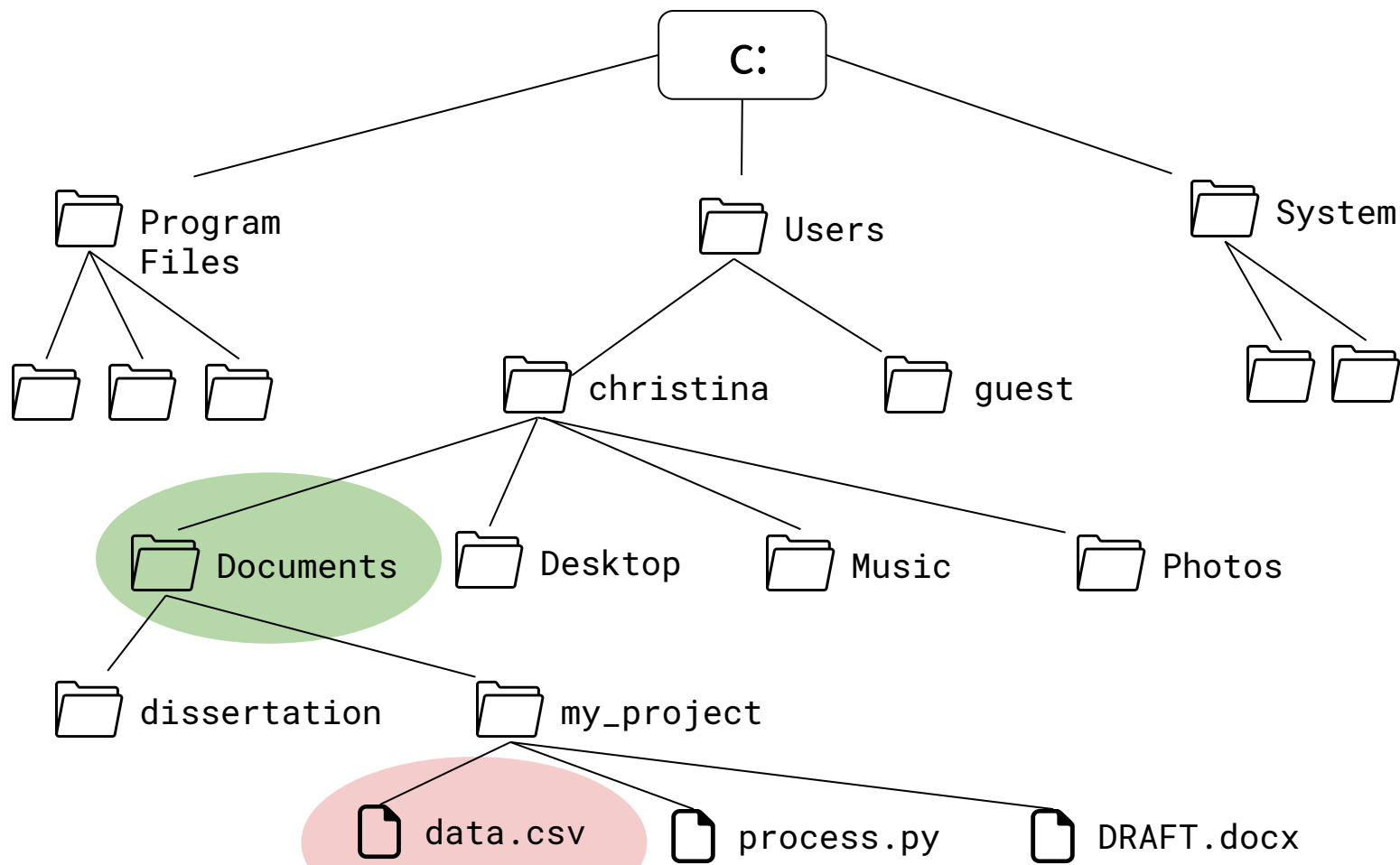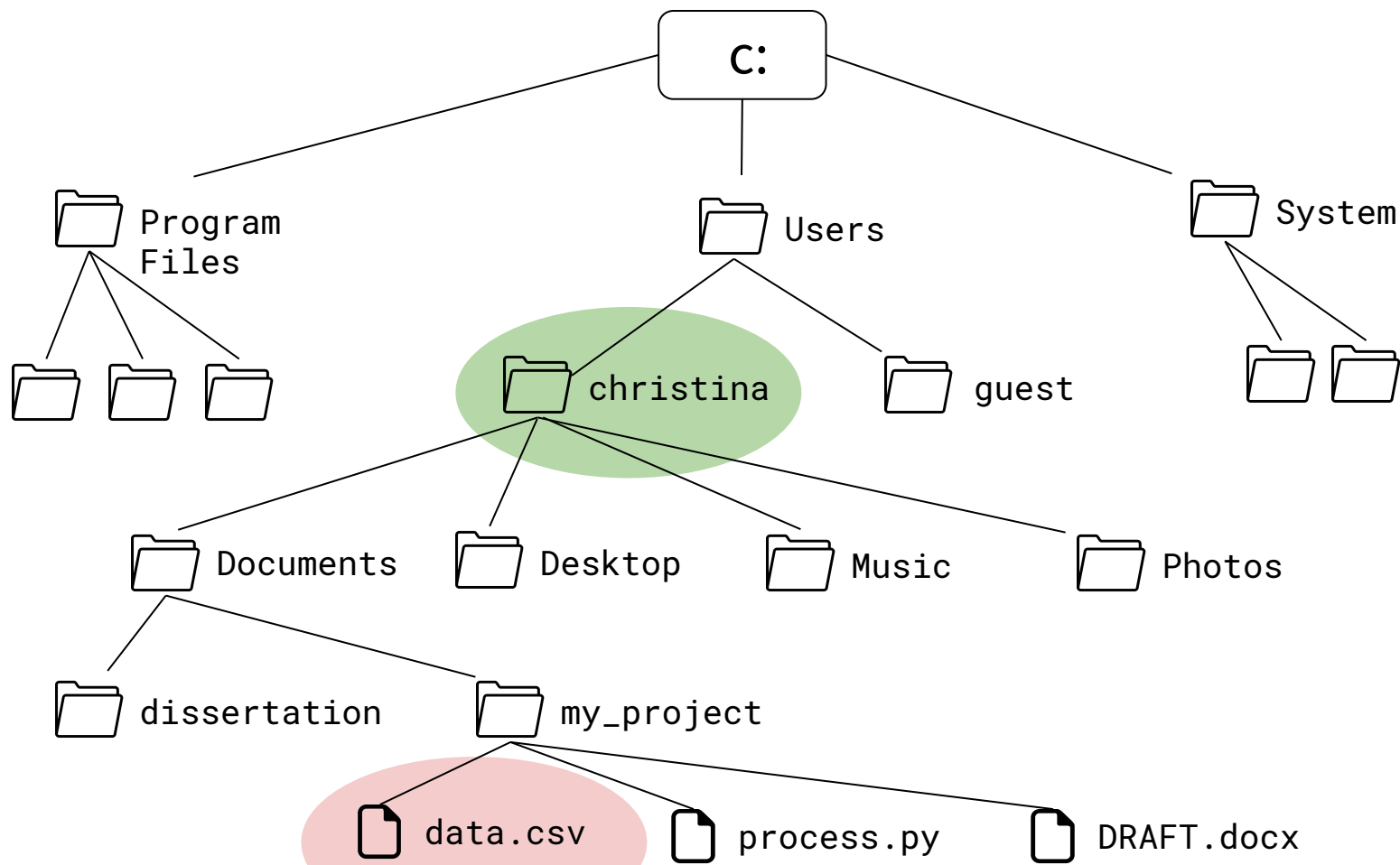
From christina: Documents/my_project/data.csv
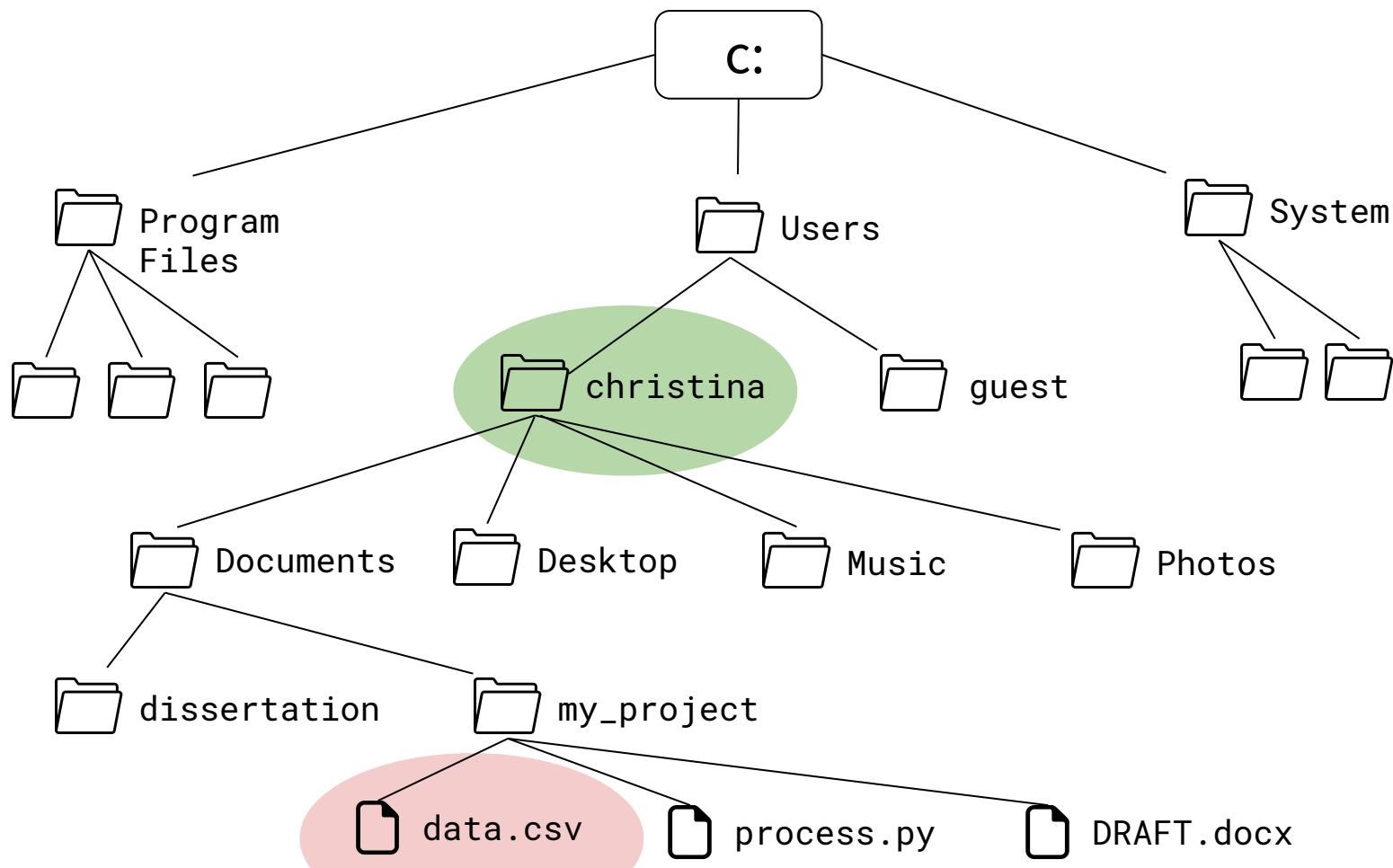
Another shortcut is .. which goes up one
directory.

From Desktop: ../Documents/my_project/data.csv
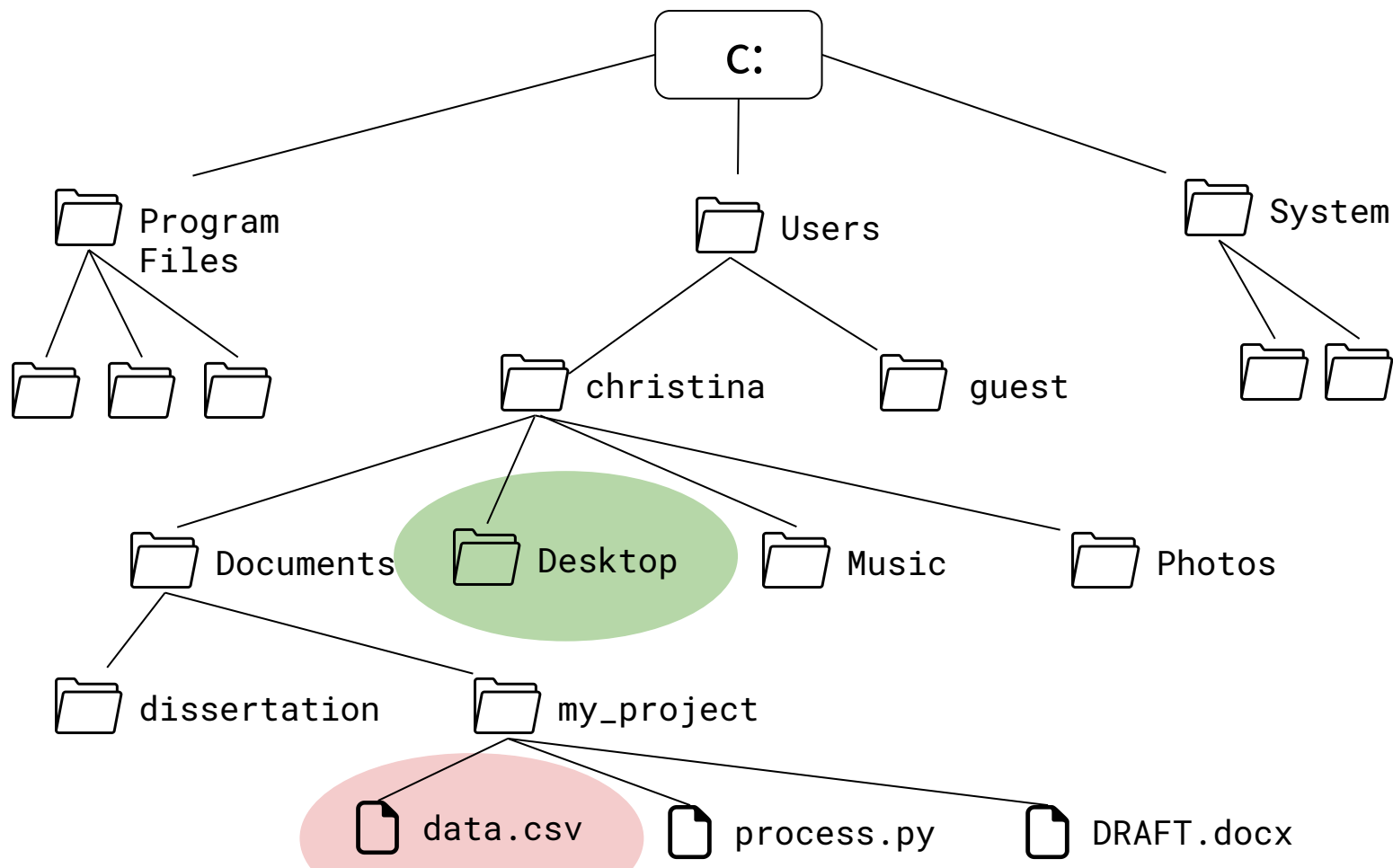
From dissertation:
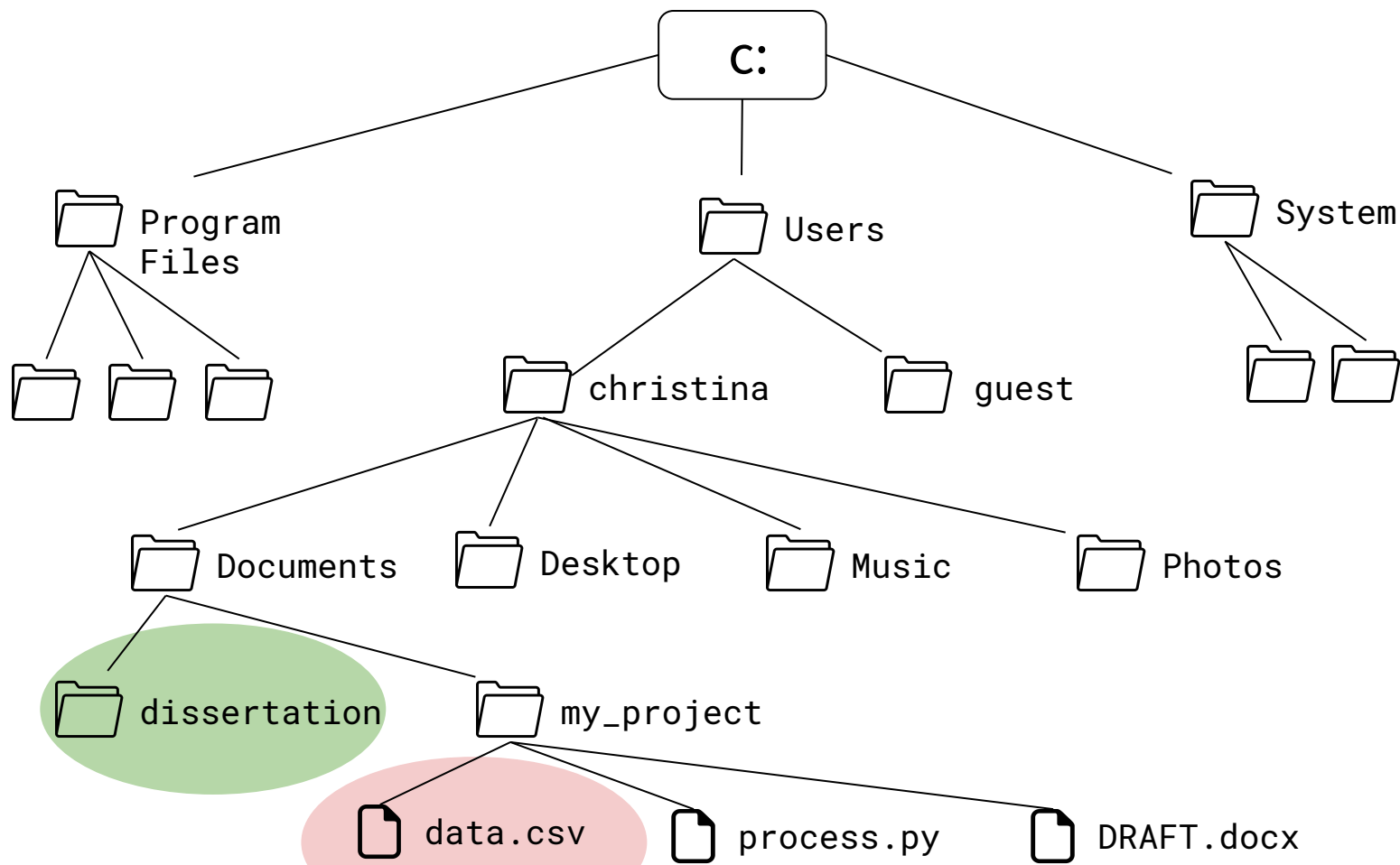
From dissertation: `../my_project/data.csv`
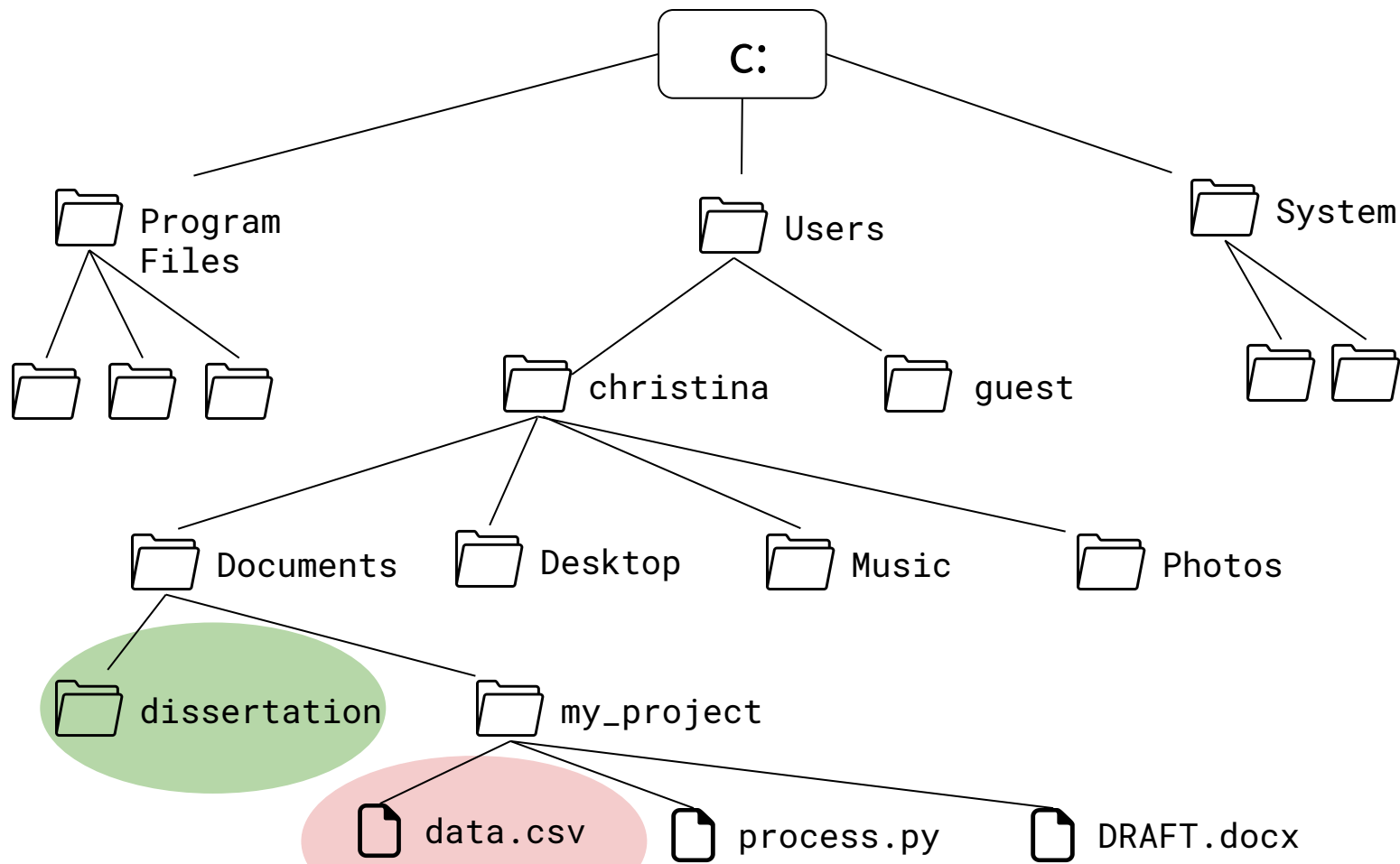
But what is the default Working Directory?

Python:

- **Where you start Python from**
- **Where you call a Python script from**

R

- Default: home directory
- RStudio Projects: the folder you associate with the project

You can always set or change the working directory

Store Project Files Together

project_directory ← working directory

➢ code
- process_data.r
- make_graphs.r
- estimate_model.r
➢ data
- people.csv
- exam1.csv
➢ output
- impact_plot.pdf
➢ reports
- chapter1.docx

msms

doc
ms-analysis.html

data

src
makefile
ms-analysis.c

bin
parse-sqt.py
ms-analysis

results
notebook.html

paper
makefile
msms.tex
msms.pdf

2009-01-14

2009-01-15
runall
summarize

2009-01-23
runall

yeast
README
yeast.sqt
yeast.ms2

worm
README
worm.sqt
worm.ms2

split1

split2

split3
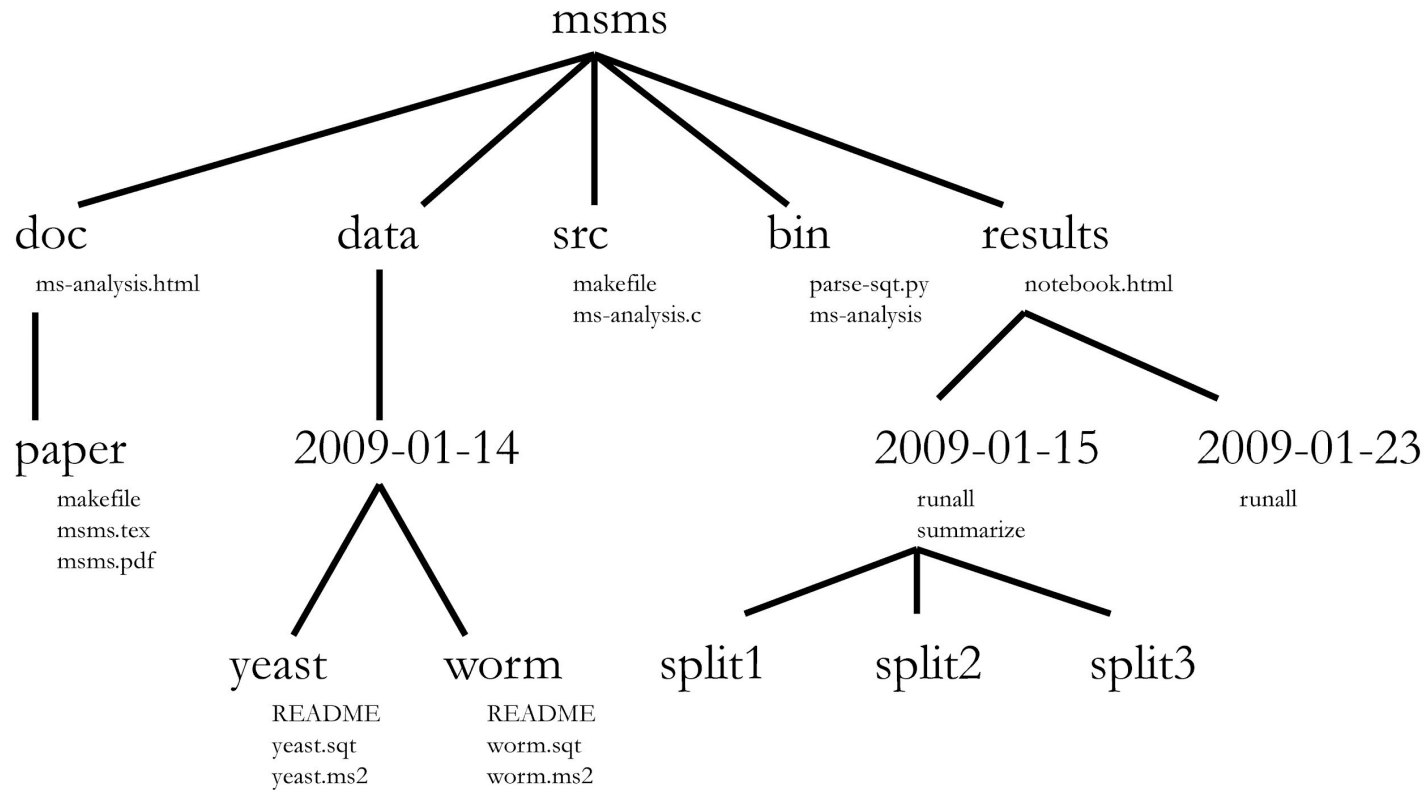
A Quick Guide to Organizing Computational Biology Projects
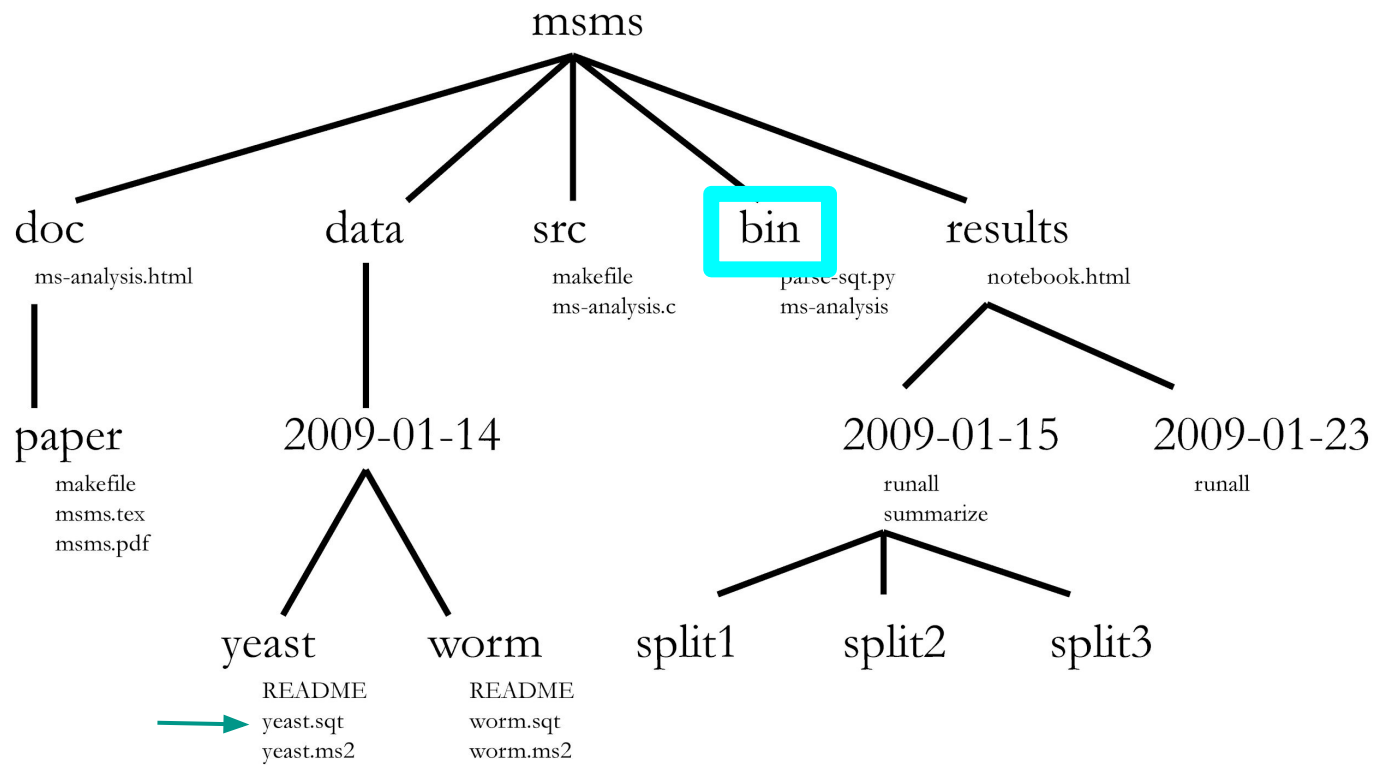
A) data/2009-01-14/yeast/yeast.sqt
B) ../data/2009-01-14/yeast/yeast.sqt
C) ~/data/2009-01-14/yeast/yeast.sqt

Files

Reminder: If my internet goes out during this workshop, that means you get a 10 minute break! You may have to log back into Zoom. Sorry!

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

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 i

[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/)
```

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

File: ammonia.pdb

```
1   %PDF-1.5
2   %-'≈ÿ
3   13 0 obj
4   <<
5   /Length 1063
6   /Filter /FlateDecode
7   >>
8   stream
9   x‚/≠VYo‹6¿~˜ØPÛD¿¿#¿¿•æ5í¿¥pÉ†›¢¿u¿-¿Ω+D+.t'˥øÔêC≠•mú¿Õæ,Ø˜Ê˙fFØwWØfiÚ4bí
10  ôÛhw¿)¿)ñ"T®hWG¿ë¿˥®„D®åÏb¡»√    6åò!¸{˜À´∑,èJZÊ‹w¢iîpEUY†‰¿{‹‚¿û¿AjD¿íá1/‹
11  ¸ôòë¿èõÆjß‚¿|è»ú-ê¿ßLÅQ¬¿-Ö@˜¿Œ"¿#‡©¿¶lÒè^[Ø»∂Fw¿∏Ixù0FÀ¸˙
12  ‡ü¡Å«h^¶˘¶∏Ô¶£Èõ͈õbæ9Ẽ^W#Z¿•¥¸
13  ∏Ç˜ªdÁêÏ¸Öî¿/√¿y)…?qñ¿›7˙∂=g´$'|}¿ÚË7¶¿¿†ƒØ?,~xb˜M±∫˚q"≥å,y0¿¿Ø¿¬pß'˚˙£©Òx¥˜?´m¸´1NòÁ¿$••˜
    çÕ~9üÚ(+¿öe¿}ª¥HÃΔW^h¿‹Iëå†Üë¿◊_"ΔË¿Ÿí=F6+RÍj )˜'¿£{õìVw˙IÔQ¿B€¿fi∑å¿¥s¿=«ª!TI'Ë¿OV≈¿¿Ü©r¡8
    ¿˚Pò\,È¿å,JQ)ãïeÔ¸∏æfi™ÅÕ/"e4/Úïò̓?˚G0AãT≠˥ÿŒ|Nè(h*≥ïÿx-!pMW7ï¿Cl∫¿5‰g;¿G%Yí›¡"Ù´"®{≥f¿‰./
    hûghC›‹§Bòfi`ÔcŒÒz{DÜ¿õa¿ùM∑«˜Ÿ$|¿Ï∂}  ÀMö•ÜÔÈV¸JA%¿Á^♣√VißTÊÁ"¿∞@æÑß¿t%◊MÚ±ÙWm¿˜
14  }wÊü ¿!§‡fi}Êÿzxé˙6Ã]fiH¿?˙µ¿*θ¿
    ∑%¿V˜›^˙¿≠≠¿í§Tdeò#‰æfi≈¿ös;˚Dµ√IWffiËÆΔEgÓq–6ùAk*ÁŒÏ¿¿æKµ_ò¿˥Ò'Œ¿lÔûsv¿≤ï¥`±è:¿YaBv[π¿T¿|Œ]3
    €¥˚˙©ö$¿Øg (¿˙Ø"?¿¿©/EJ9¿Å7gj¿‹I¿¿≠˙±E63e‚'"AáÓ8¿Á[˚∂}mzflÈ¿π∂ÆW›c{¿¿÷CÃmÕ∏ë©œTflt
15  ≈|¿'f8ØôCcà¿1¿ãŒôø(}…
16   3Õ¸¸¢ü˜÷b±ƒ\=Ø9ÿo¿DWœl¿ÒÇiåãSo˜Ω>ûßH˚¿Ê§@0.ü§#'Gg¡‰∫]ËêãaÏΔBfl€©´qÑ∫K◊˜∞í~xÊÁCï∫o¿/Ú[hè]
    Äx˙θ∫ ‚'Œ¿ü"¿ÿ"¿Ñƒ¶êî‚'NÓK·íV8èEQPëóOÁÔE›1¿æ0¿_8¿ß"`jßú)r;ç˜¿¢˜ÄΣ¿¿¿¿∑"#¿Øü¶
    í˚o¿v¿Ú¿¿m¿¿¿k¿¿^>¥aã¸¿»B;w¿'C3å¸?>æ˙ƒ¿Ñ˚    ♣Yä¿}}1L∑/6Íîót*Ò%u∫
    Î•N¿CCŒ¿N"‡Y≤•/S4¿Á∑ûoW?ÌÆ¸¿¥¿
17  endstream
18  endobj
```

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,,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 R and Python let you write plain text files.

Stand alone options:

https://workshops.rcs.northwestern.edu/install/texteditor/

Time to Review!

Data Types

Numbers

Integers

-38291423
0
2

Decimal/Float

3.0
-432.2343253
4.938e-10

Character

AKA: text, string

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."

"whitespace"

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"	"110 cats"	"110 cats"
"3 cats"	"3 cats"	"3 cats"
"apple"	"Apple"	"apple"
"mushroom"	"Mushroom"	"Apple"
"Apple"	"apple"	"mushroom"
"Mushroom"	"mushroom"	"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

produces substrings

Joining Strings: Concatenate

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

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

`paste("Red", "bull", sep=" ") = "Red bull"`

`paste("Red", "bull", sep="") = "Redbull"`

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) =
```

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
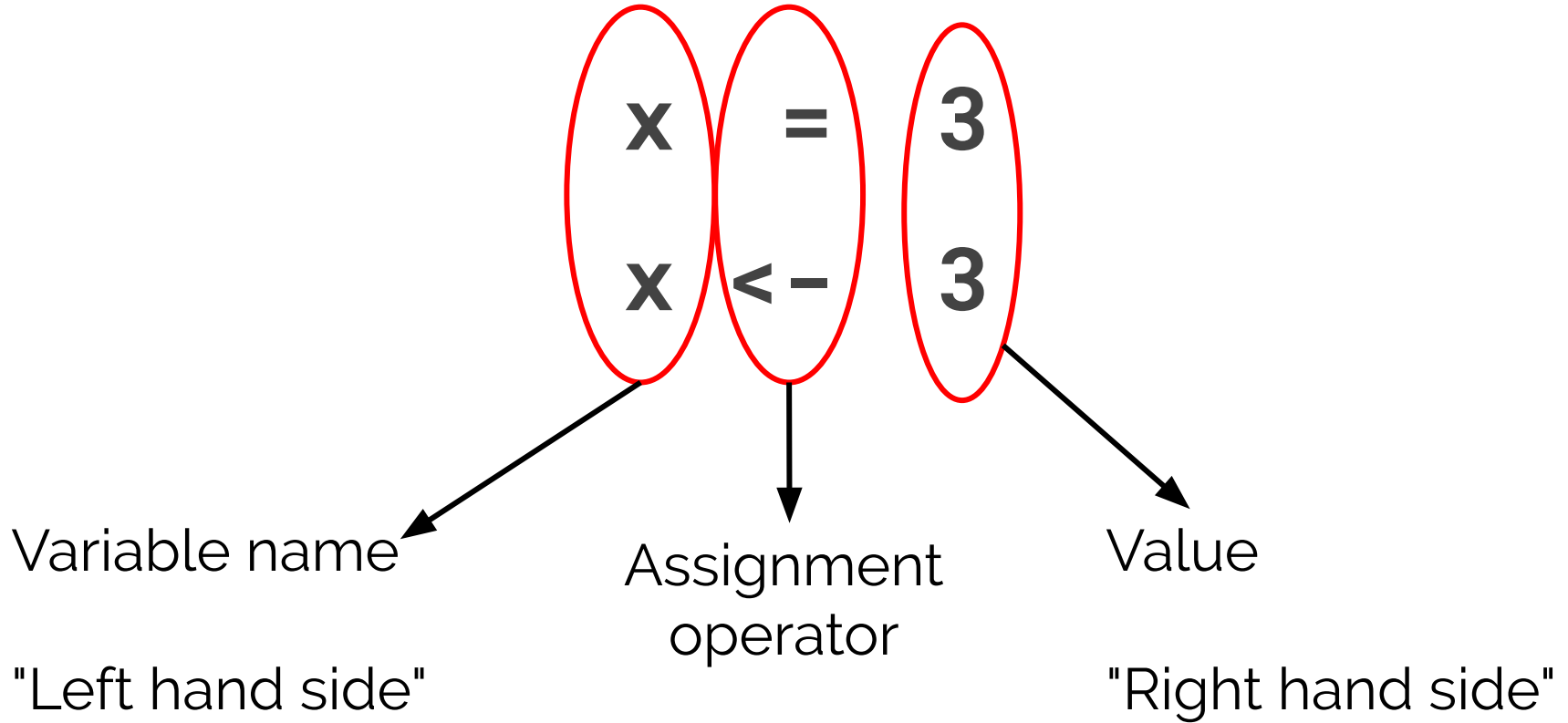
Variables

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

$$x = 3$$

$$y = 5$$

$$x = x + y$$

$$x$$

x is 8

Variables

$$x = 3$$

$$y = 5$$

$$x = x + y$$

$$y = 7$$

$$x$$

x is 8

Variable Names

Case matters

Start with a letter

Make names meaningful

Style conventions

    camelCase

    separate_with_underscores

Lists, vectors, arrays

Multiple Related Values

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]

      0   1   2   3   4   5   6   7

R vector

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

      1   2   3   4   5   6   7   8

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

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

Empty list or vector

[]

c()

List Indexing (Python)

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

30

List Indexing (Python)

```python
ages = [42, 30, 24, 24, 27, 35, 39, 22]
ages[1:4]
```

[30, 24, 24]   Python indexing is

exclusive of the end position

Vector Indexing (R)

```
ages <- c(42, 30, 24, 24, 27, 35, 39, 22)
ages[1:4]
```

[42, 30, 24, 24]   R indexing is *inclusive* 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**]

Prepend (R): [**50**, 42, 30, 24, 24, 27, 35, 39, 22]

Append - doesn't take a lot of work

Append - doesn't take a lot of work

Prepend - takes a lot of work

Prepend - takes a lot of work

Prepend - takes a lot of work

Prepend - takes a lot of work

Prepend - takes a lot of work

Prepend - takes a lot of work

Prepend - takes a
lot of work

Time to Review!

Conditions

Conditions

Expressions that produce a boolean value:

True or False

Comparisons

>   <   >=   <=

==   !=

Comparisons

age = 13   Assignment

age < 21   Comparisons

age == 14

Compound Conditions

first_age = 13

second_age = 15

first_age < 21 or second_age < 21

Compound Conditions

```
first_age = 13
```

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

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

Exercise: Two Truths and a Lie

Open the exercise from the folder you downloaded.

Complete the top of the page first, and then write your three statements. (6 minutes)

Then, I will assign you to breakout rooms to share your Two Truths and a Lie with other participants. Share your screens one at a time and try to guess which of their statements are the lies. (7 minutes)

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                    18, 19, 20, 21...

    print "Go Vote!"

else if age >= 16               16, 17

    print "Learn to drive"

else                            15, 14, 13, 12,
                                11, 10...
    print "Too young!"
```

If Statements: Multiple Actions

```
if age >= 18

    print "Go Vote!"

else if age >= 16

    print "Learn to drive"

    print "Stay in school"

else

    print "Too young!"
```

If Statements: Syntax

```python
if age >= 18:

    print("Go Vote!")

elif age >= 16:

    print("Learn to drive")

    print("Stay in school")

else:

    print("Too young!")
```

```
if (age >= 18) {

    print("Go Vote!")

} else if (age >= 16) {

    print("Learn to drive")

    print("Stay in school")

} else {

    print("Too young!")

}
```

Writing Pseudocode
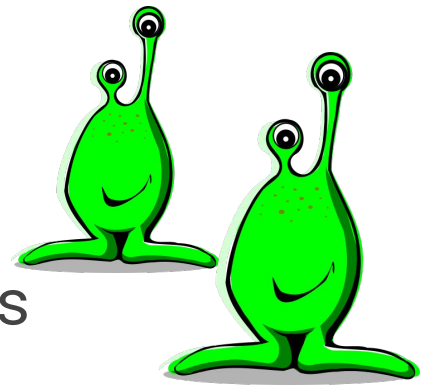
```
if color is green and has eye stalks

    send to Mars

else

    send to Earth
```
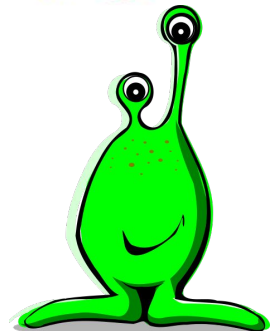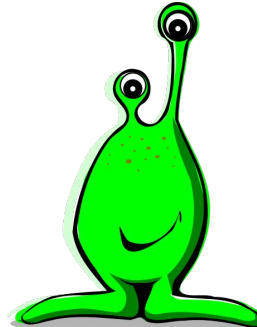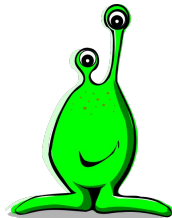
Exercise

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

Group 1
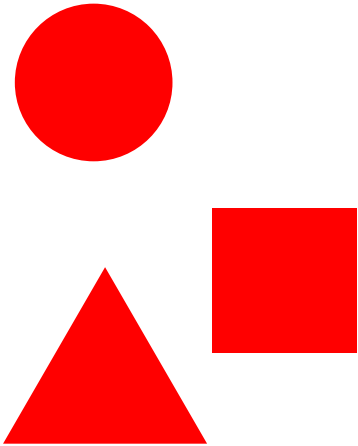
Group 2
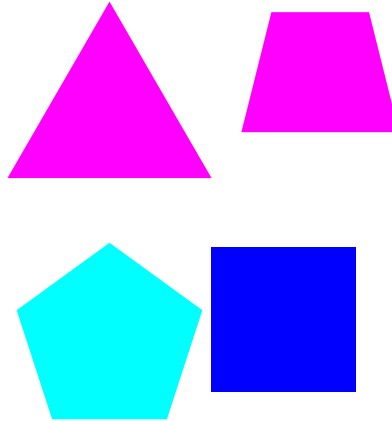
Group 3
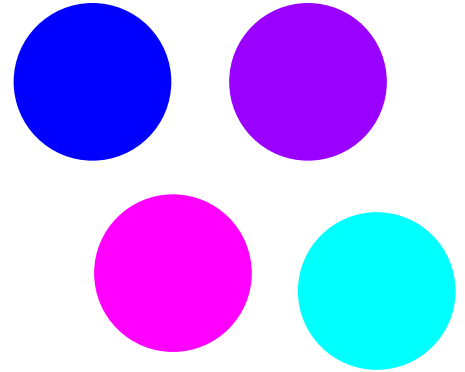
Exercise

```
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/vector of values*

    do something

For Loop

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

Variable called age

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

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

```
for dog in row

bark(dog)
```

Arf!

```
for dog in row
bark(dog)
```

Hav!

```
for dog in row
    bark(dog)
```

for dog in row

bark(dog)

Voff!

for dog in row

bark(dog)

Mung!

for dog in row

bark(dog)

Yip!

For Loop

```
total = 0

for i in 1:10

   total = total + i

print total
```

1, 2, 3, 4, 5, 6, 7, 8, 9, 10
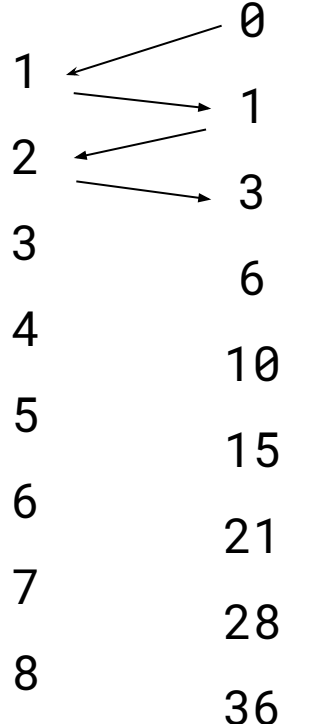
For Loop

```
total = 0

for i in 1:8

    total = total + i

print total
```

i

total

1
2
3
4
5
6
7
8

0
1
3
6
10
15
21
28
36

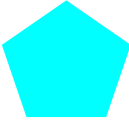# For Loop

```
total = 0

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


for i in ages

    total = total + i

print total
```
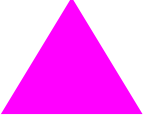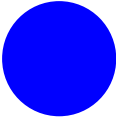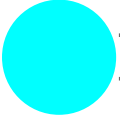
i

total

42

0

30

42

24

42

24

72

27

96

35

120

39

147

22

182

221

243

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
```

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

Arf!

```
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)
```

Functions

Functions

- Take input values
- Can return an output value
- Python and R functions usually don't alter their input values (but they sometimes do!)

Functions

The *len()* function in Python takes a list as input and returns the length of that list:

len([1, 2, 3, 4]) **4**

The *length()* function in R takes a vector as input and returns the length of that vector:

length(c(1, 2, 3, 4)) 4

Function Definitions

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

read.csv(file, header = TRUE, sep = ",", quote = "\"",
dec = ".", fill = TRUE, comment.char = "", ...)

Function Definitions: Function Name

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

==read.csv==(file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", ...)

Function Definitions: Parameters

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

read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", ...)

Function Definitions: Parameter Names

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

read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)

Function Definitions: Default Values

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

read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", ...)

Function Definitions: Required Parameters

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

read.csv(file, header = TRUE, sep = ",", quote = "\"", dec = ".", fill = TRUE, comment.char = "", ...)

Calling Functions: Python

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: Order Matters

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

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

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

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

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

Calling Functions: R

read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)

```
read.csv("results.txt", sep="\t")
```

```
read.csv("results.txt", "\t") ❌
```

```
read.csv("results.txt", TRUE, "\t")
```

Calling Functions: R: Positional and Named

read.csv(file, header = TRUE, sep = ",", quote = "\"",
        dec = ".", fill = TRUE, comment.char = "", ...)

```
read.csv(file="results.txt", TRUE, "\t")
```

Calling Functions: R: Positional and Named

1　　**2**
read.csv(<u>file</u>, header = TRUE, sep = ",", quote = "\"",
　　　dec = ".", fill = TRUE, comment.char = "", ...)

　　　　　　　　　　　　　1　　　**2**

read.csv(<u>file="results.txt"</u>, TRUE, "\t")

Calling Functions: R: Positional and Named

read.csv(file, <u>header = TRUE</u>, sep = ",", quote = "\"",
    dec = ".", fill = TRUE, comment.char = "", ...)

read.csv(<u>header=TRUE</u>, "results.txt", "\t")

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!")
```

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)
```

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

variables observations values

Rows: Observations

Person (or mouse, worm, etc.)

Country

Year

Run/trial of an experiment
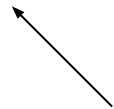
Chemical

Sample

Columns: Variables

Measurements

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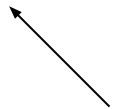> condition
> label for the observation (country name)

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

Trial 1   Trial 2

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