

Programming Concepts

with instructor Colby Witherup Wood

Workshop originally created by Christina Maimone

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

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

Goals:

3. At the very end, I will talk about which language you should learn, R or Python.

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

The screenshot shows the Microsoft Excel interface. The ribbon at the top includes 'Draw', 'Page Layout', 'Formulas', 'Data', 'Review', and 'View'. The 'Formulas' tab is active, showing the 'fx' button and the formula bar. The formula bar contains the text '=SUM(D2:I2)'. Below the formula bar, a tooltip displays the syntax 'SUM(number1, [number2], ...)'.

The spreadsheet data is as follows:

B	C	D	E	F	G	H	I	J	K	L	M	N
r_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.

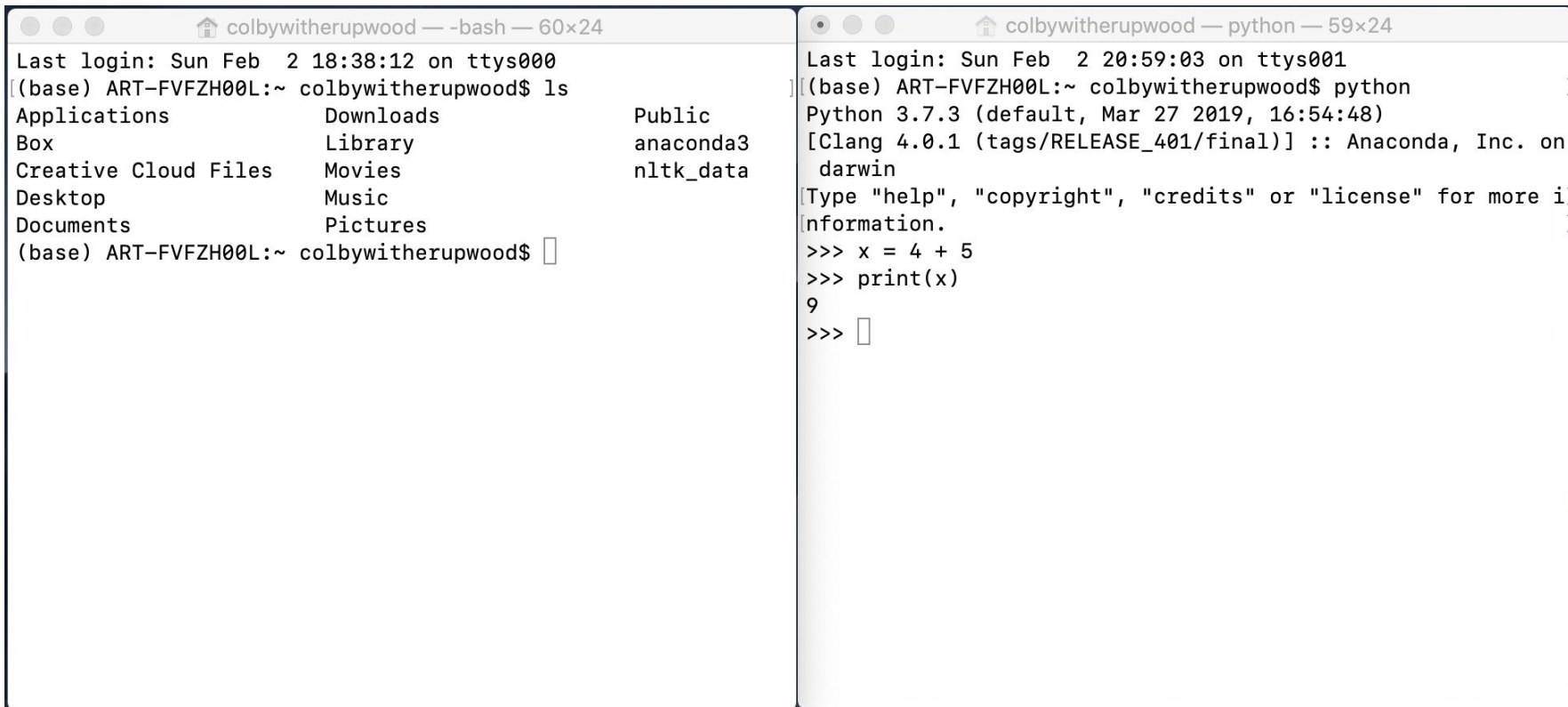
How do we talk to our computer in Python or R?

- **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)

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**. They allow both interactive and batch programming.

Command prompts



```
colbywitherupwood — -bash — 60x24
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$
```

```
colbywitherupwood — python — 59x24
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]
information.
>>> x = 4 + 5
>>> print(x)
9
>>>
```

Time to Review!

Open the ProgrammingConceptsReview document from the folder you downloaded.

Filesystems



Filesystems

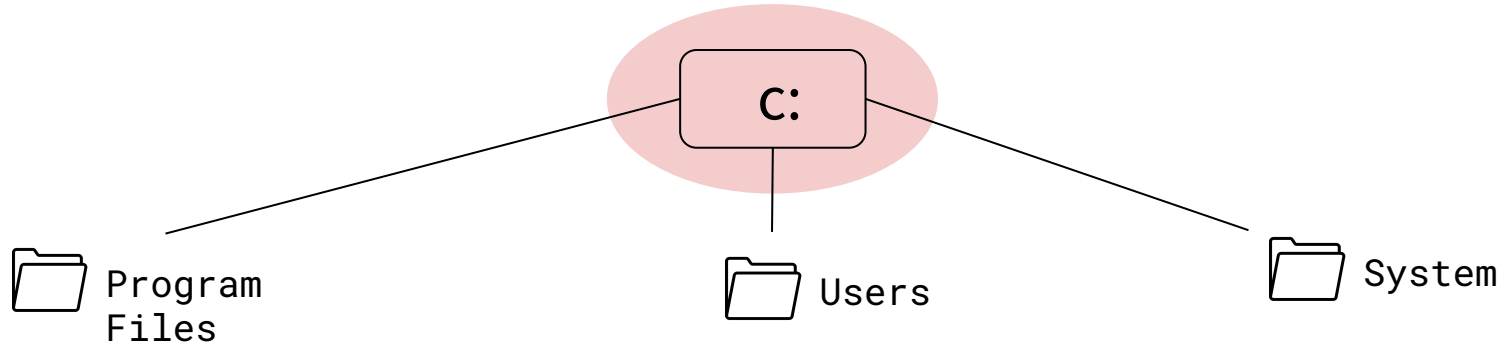
Coding requires us to move away from point and click.

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

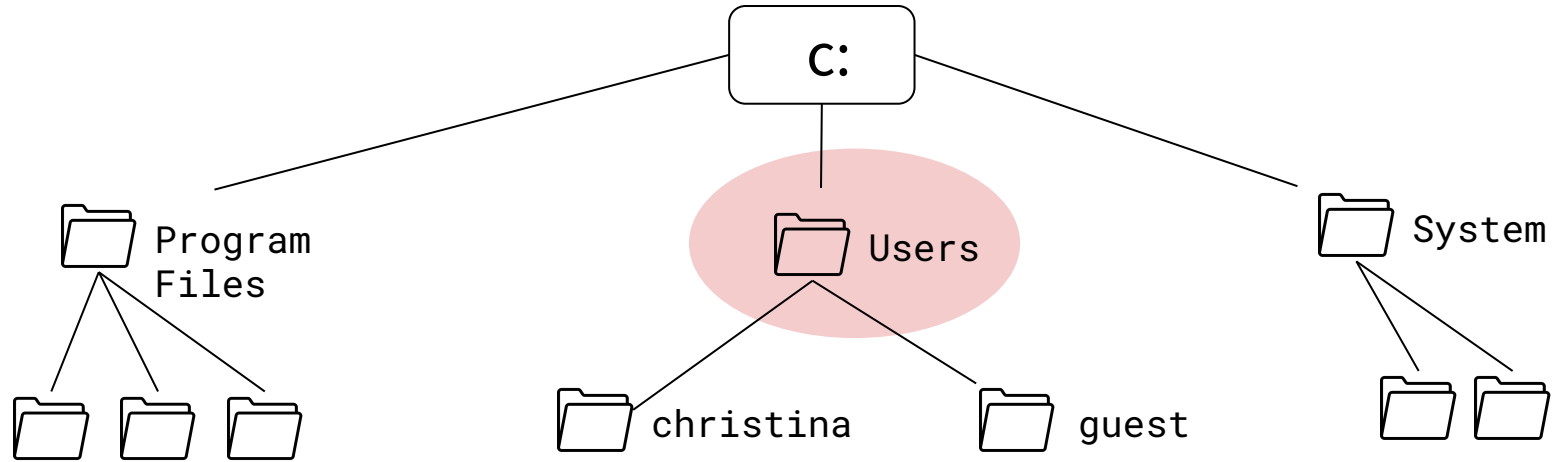
Every file has an **absolute path**, which starts with the **root**.

C:

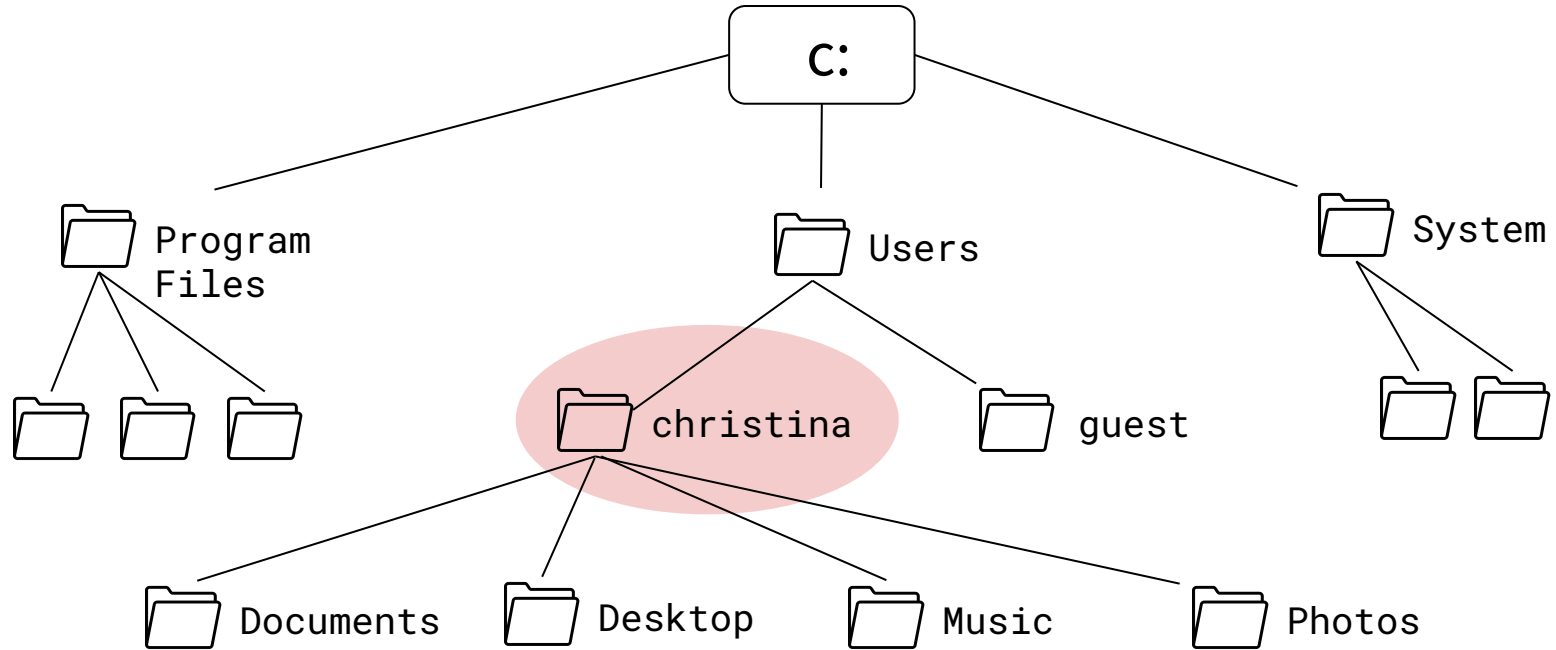
C:\



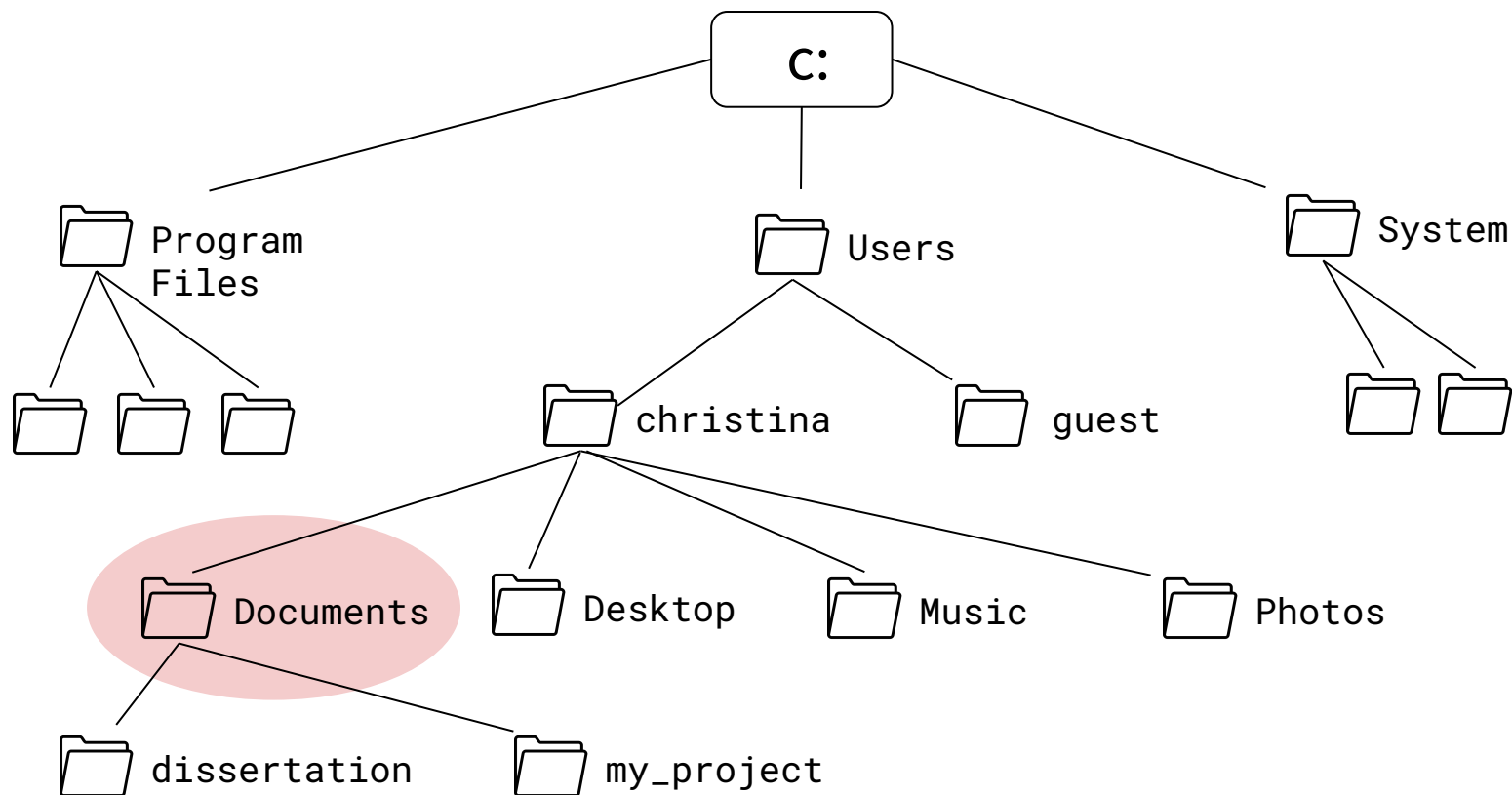
C:\Users\



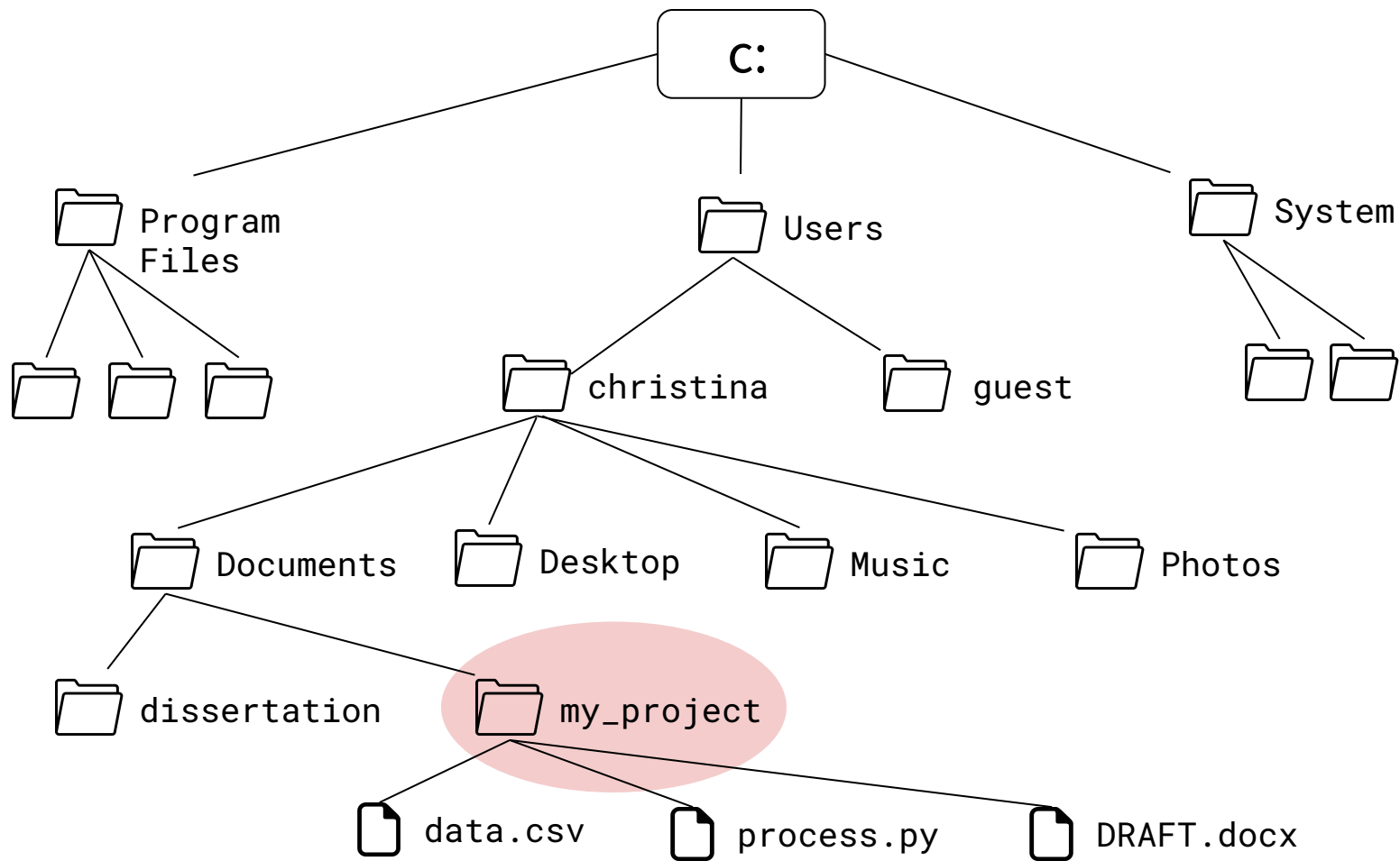
C:\Users\christina



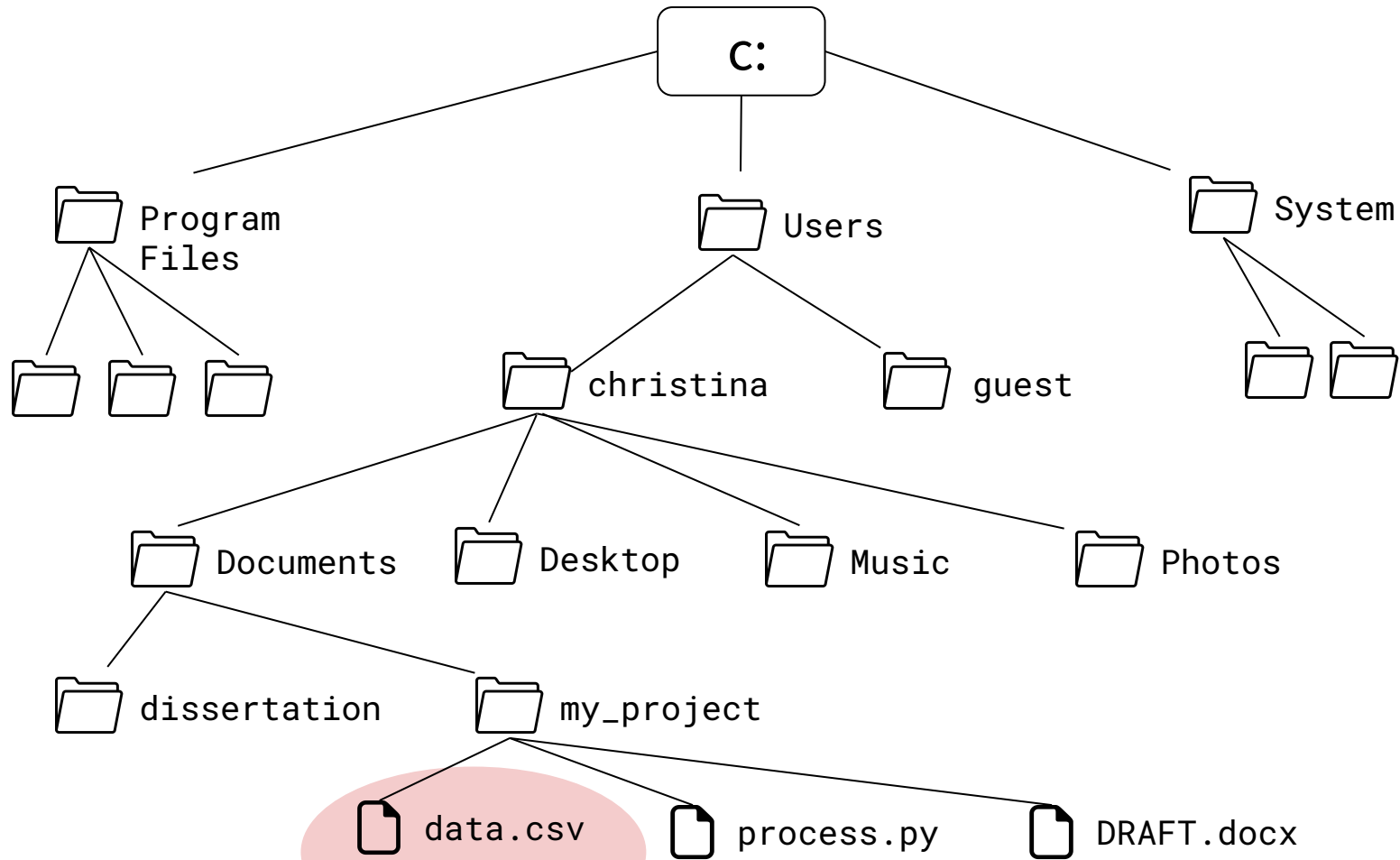
C:\Users\christina\Documents



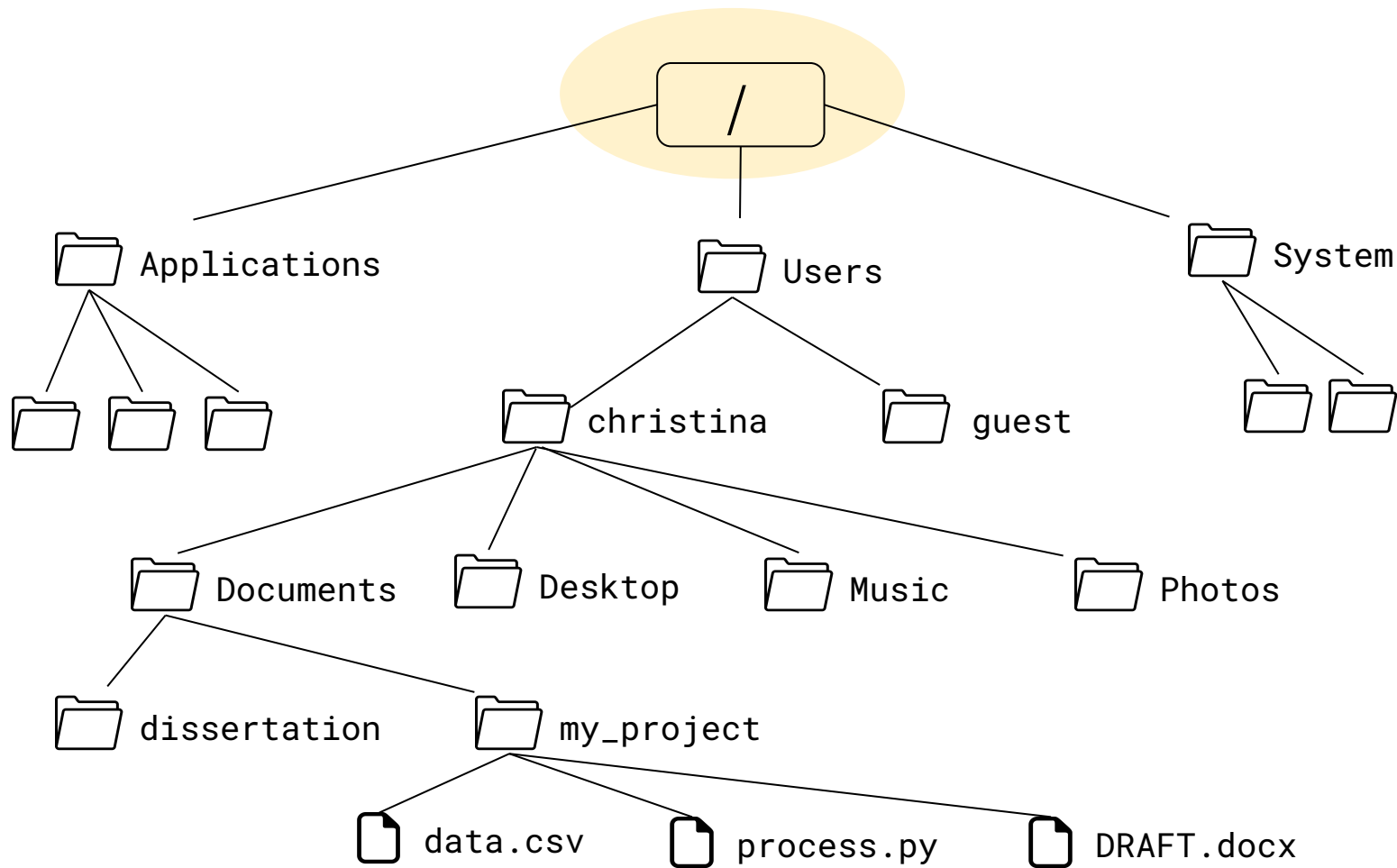
C:\Users\christina\Documents\my_project



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



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



Absolute Paths

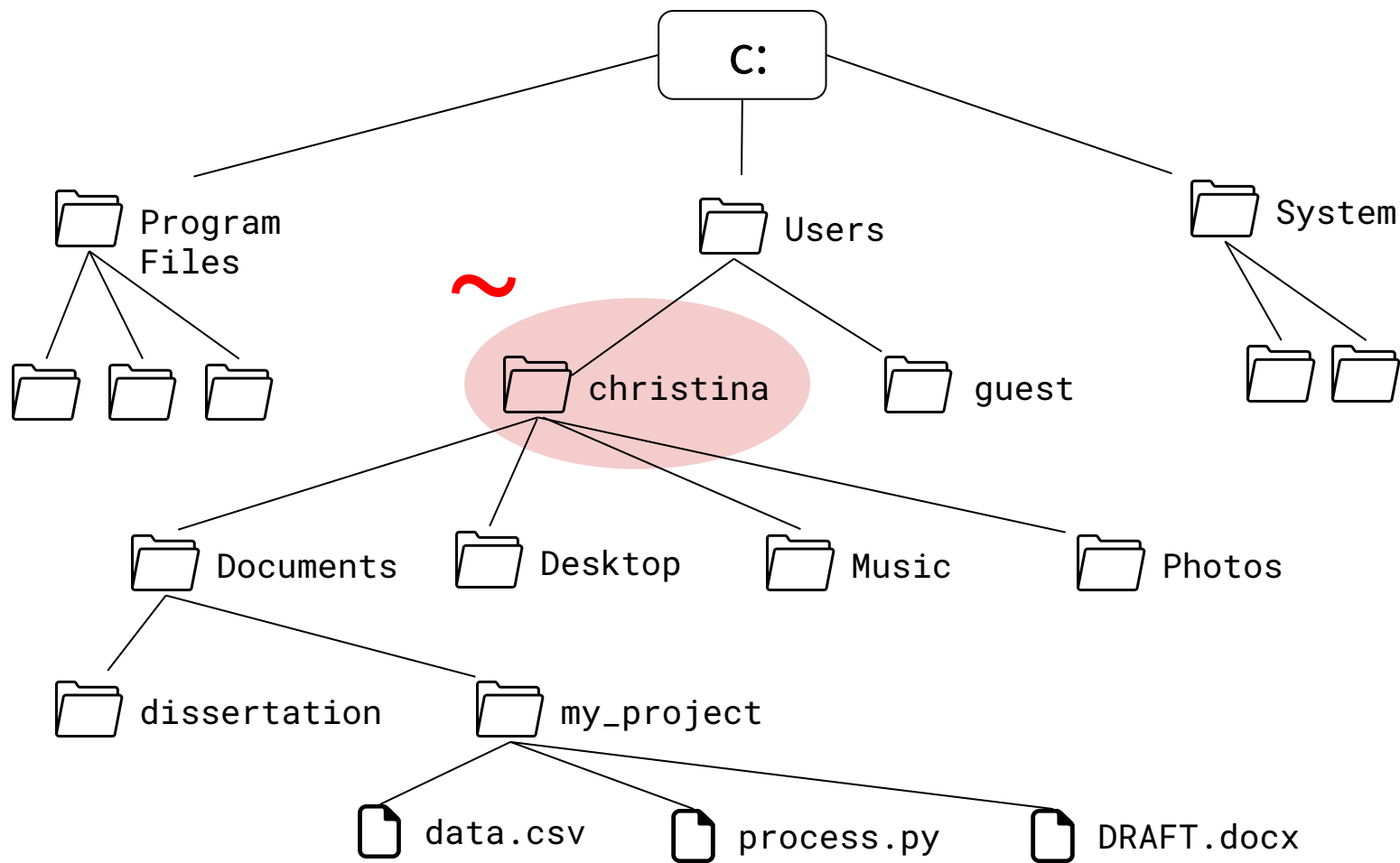
C:\Users\christina\Documents\my_project\data.csv

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

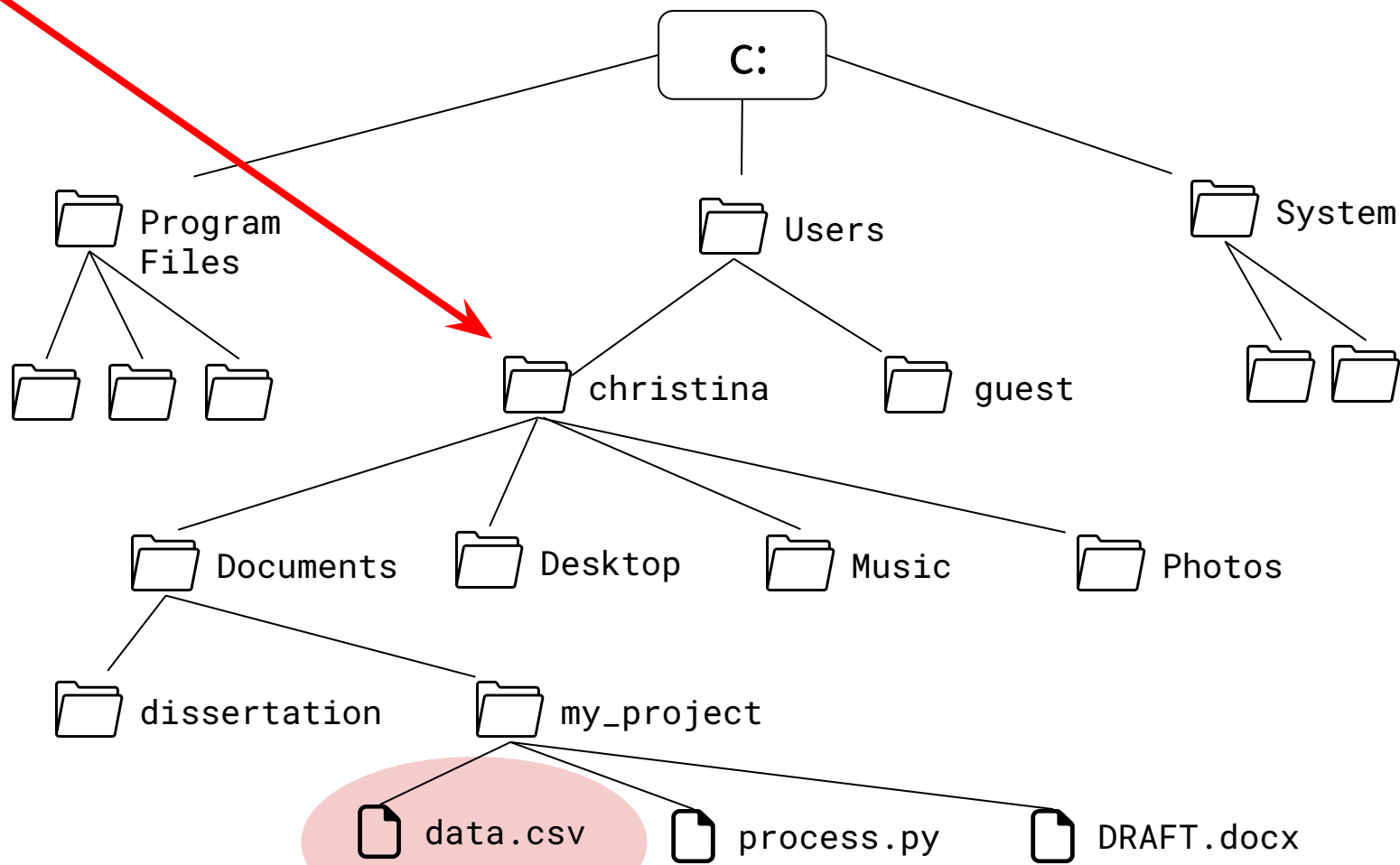
Home directory

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



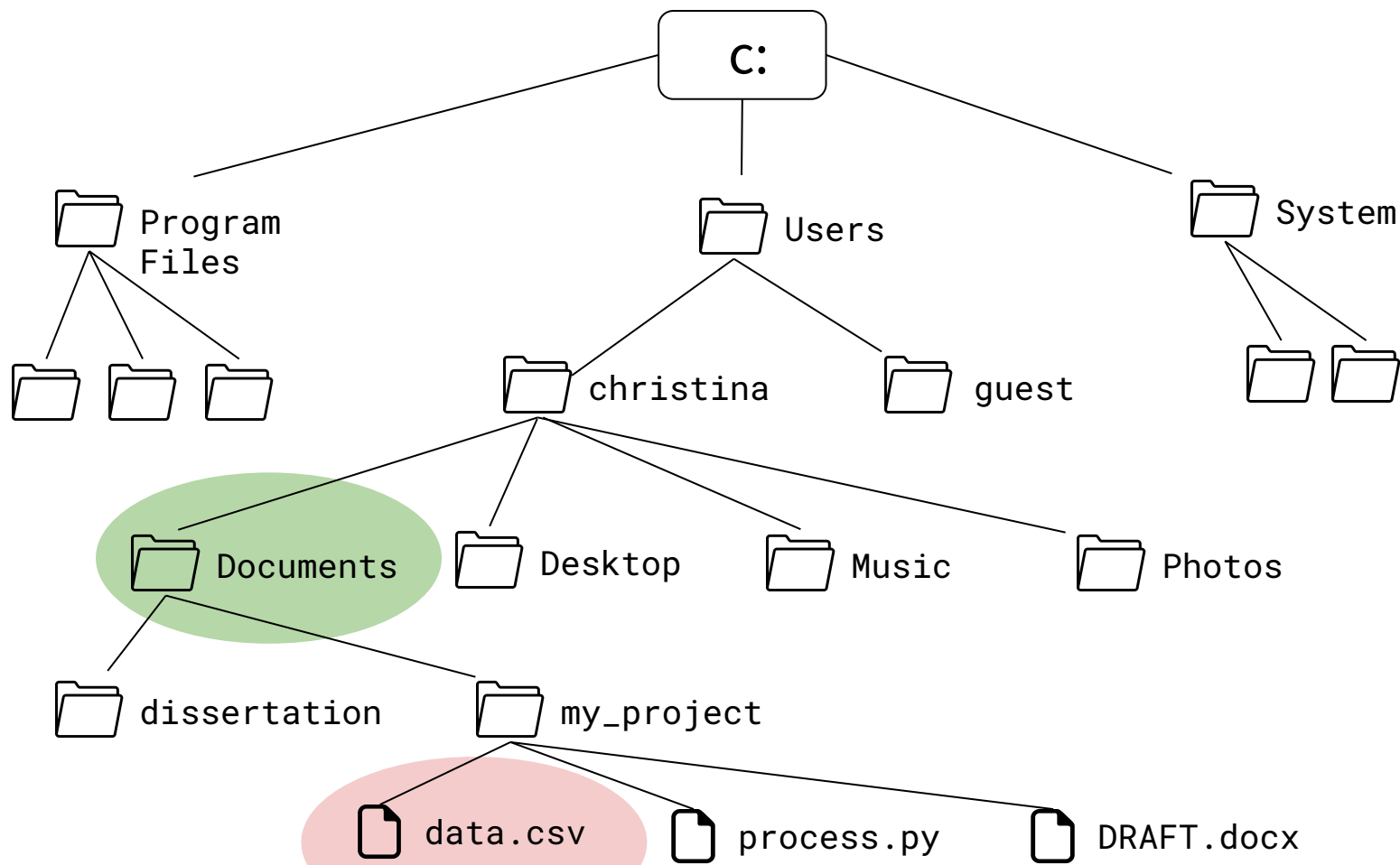
Working Directory

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

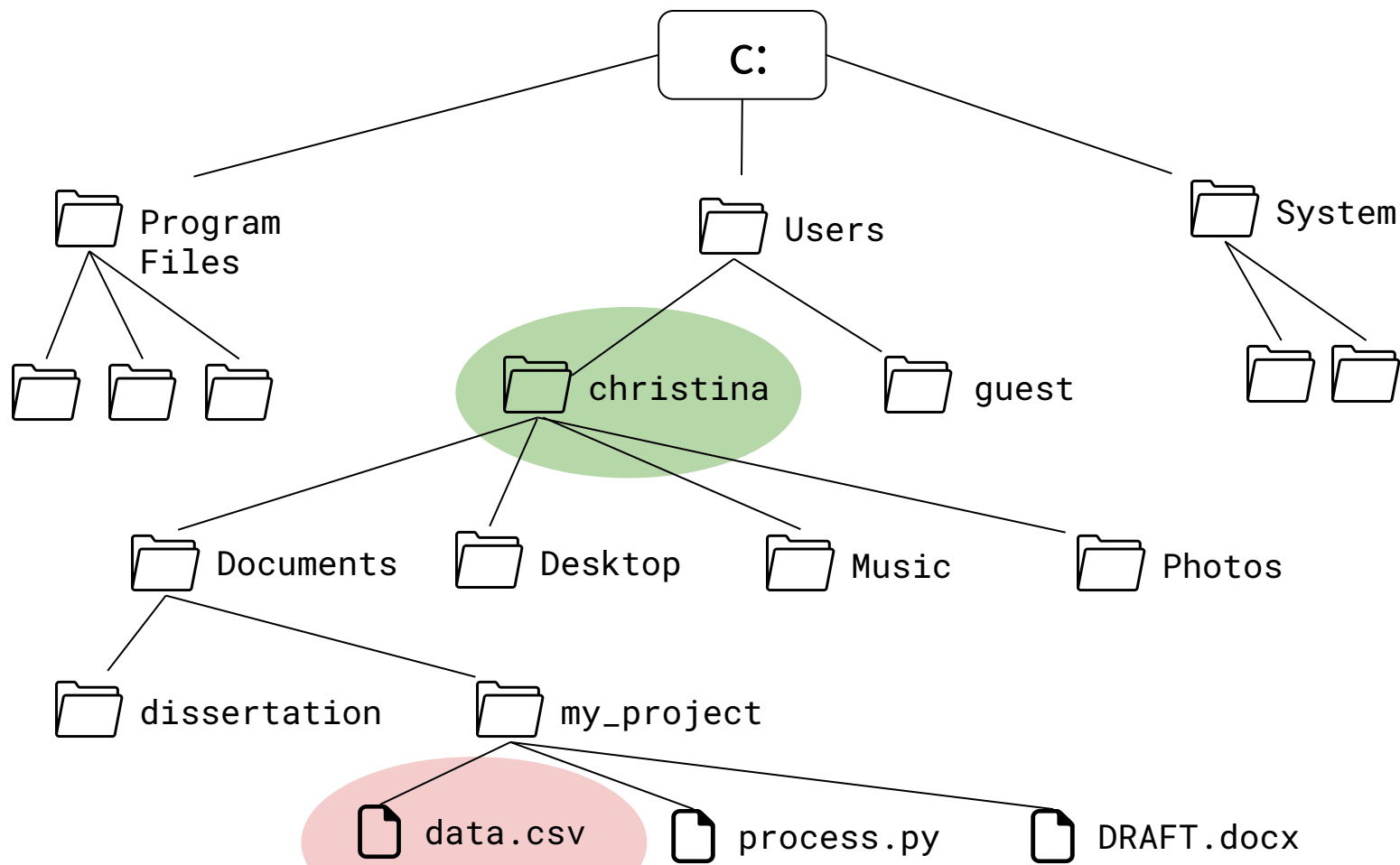
This is where the computer starts when looking for files

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

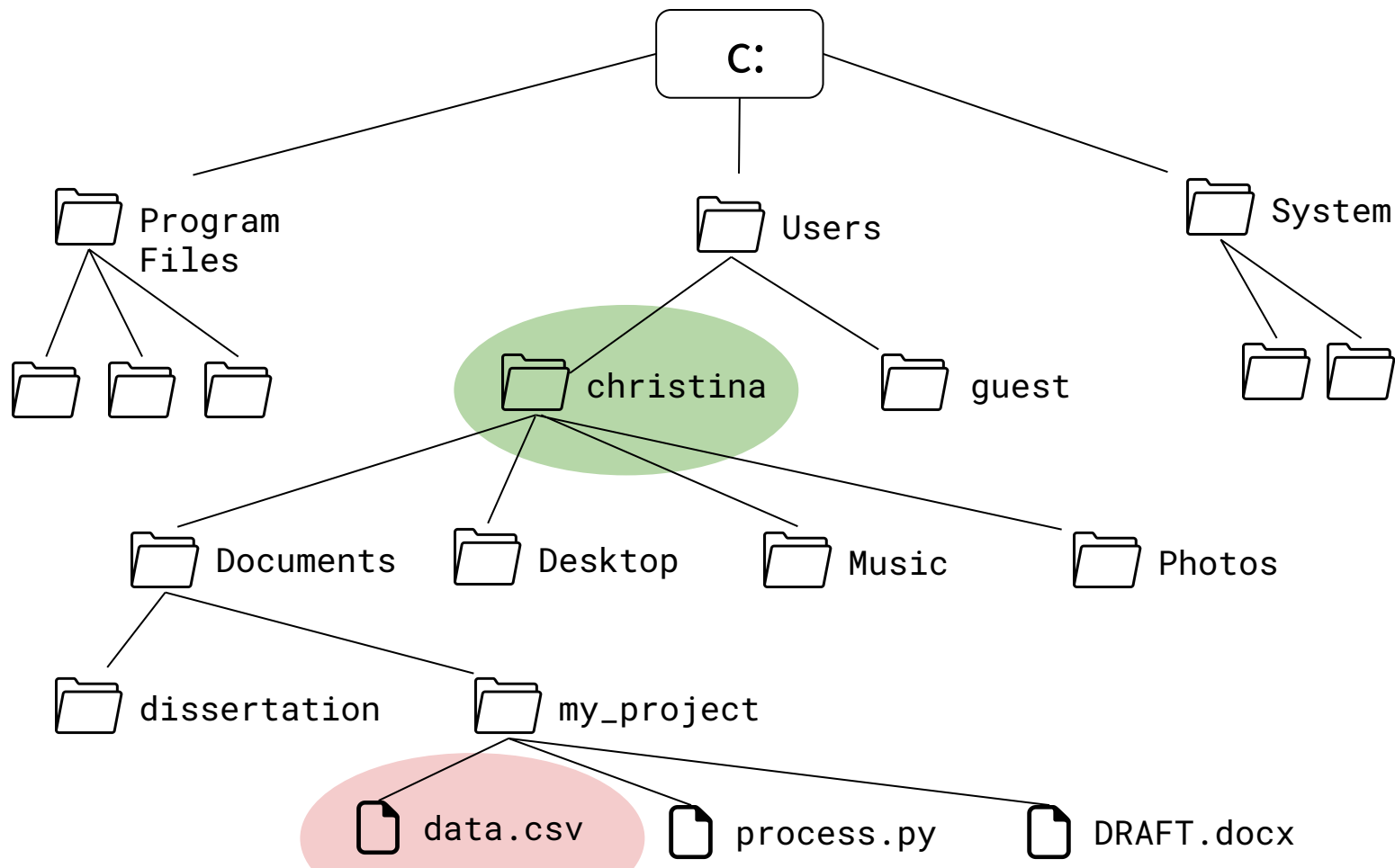
Relative Path from Documents: **my_project/data.csv**



From christina:

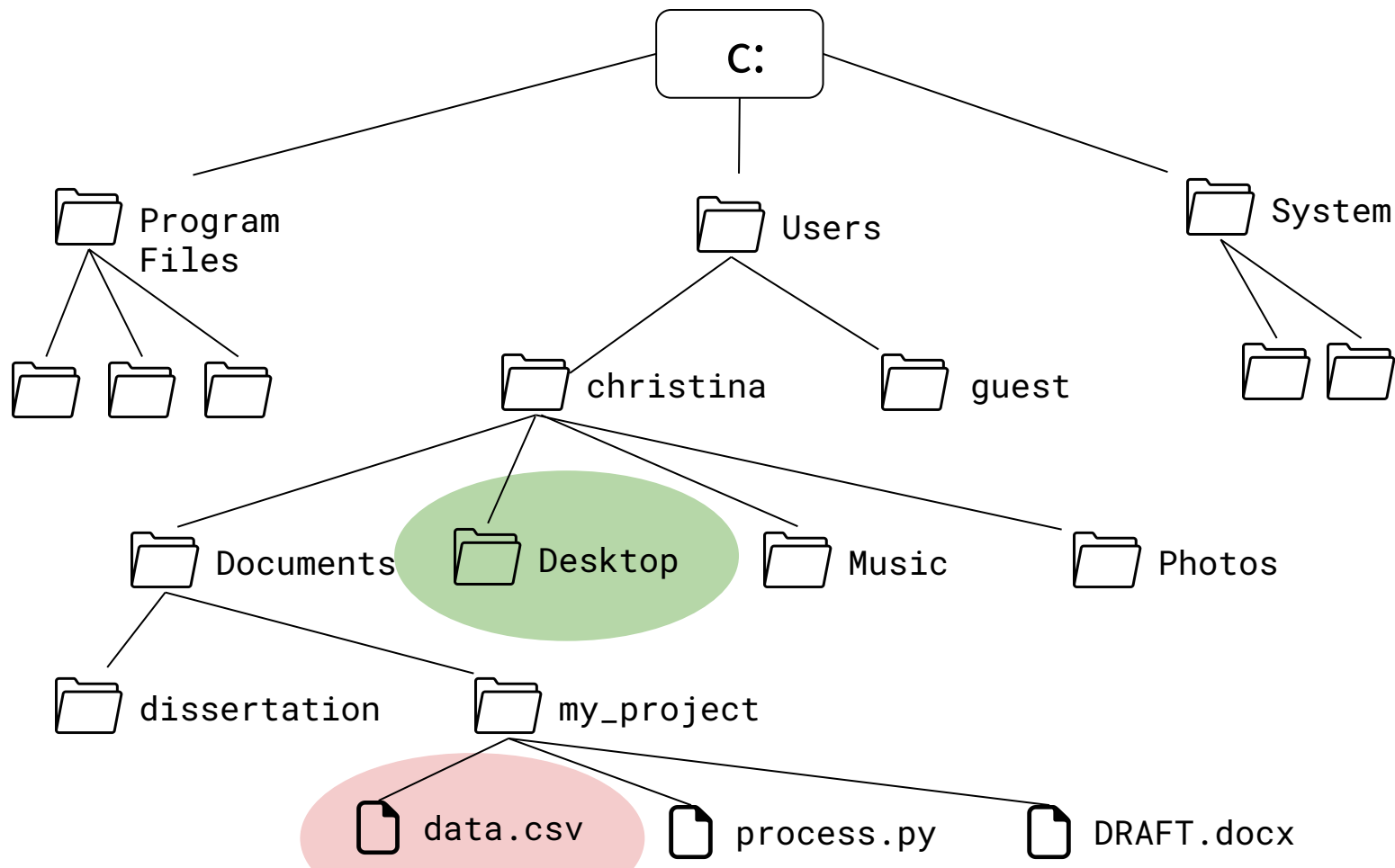


From christina: Documents/my_project/data.csv

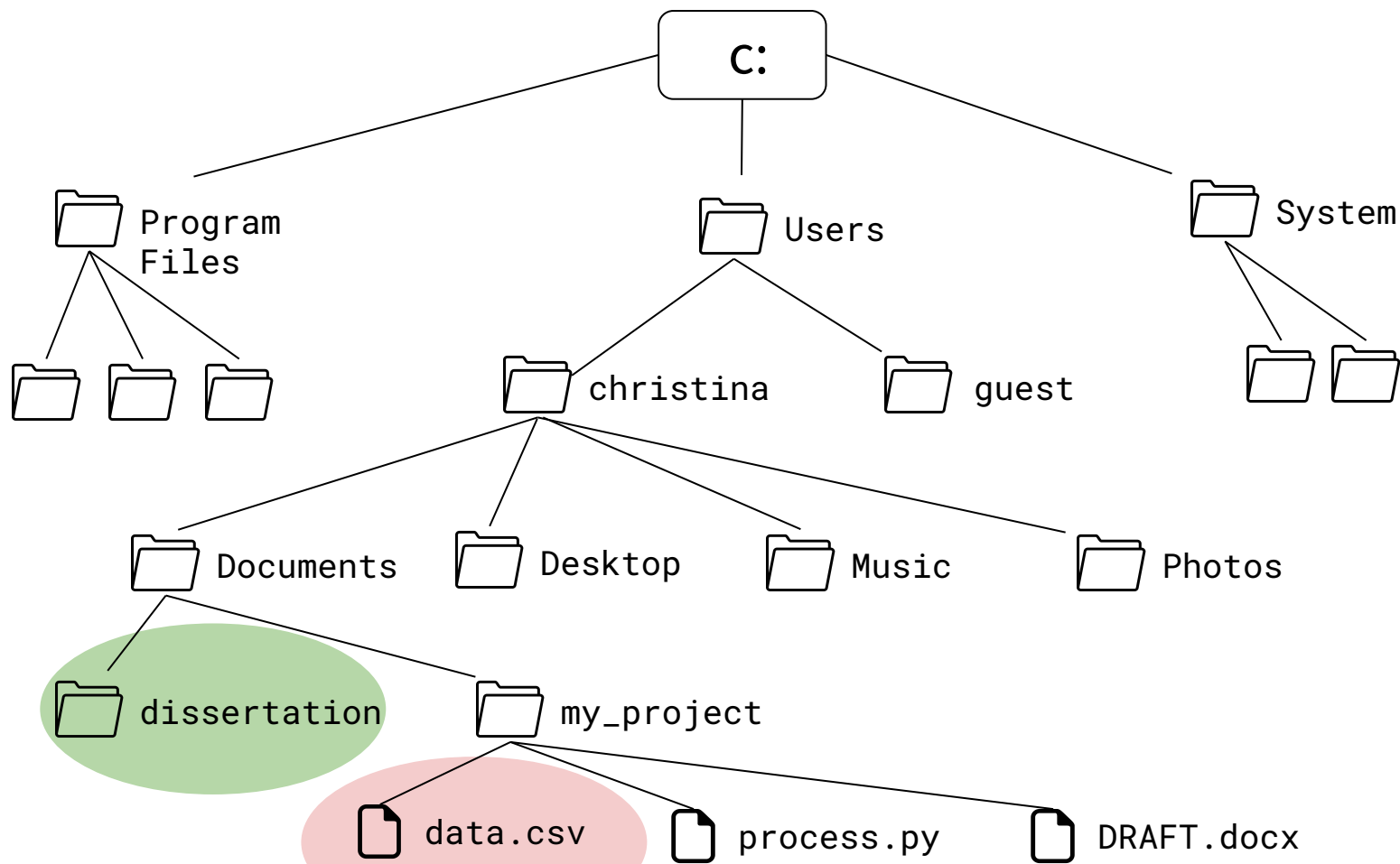


Another shortcut is `..` which goes up one directory.

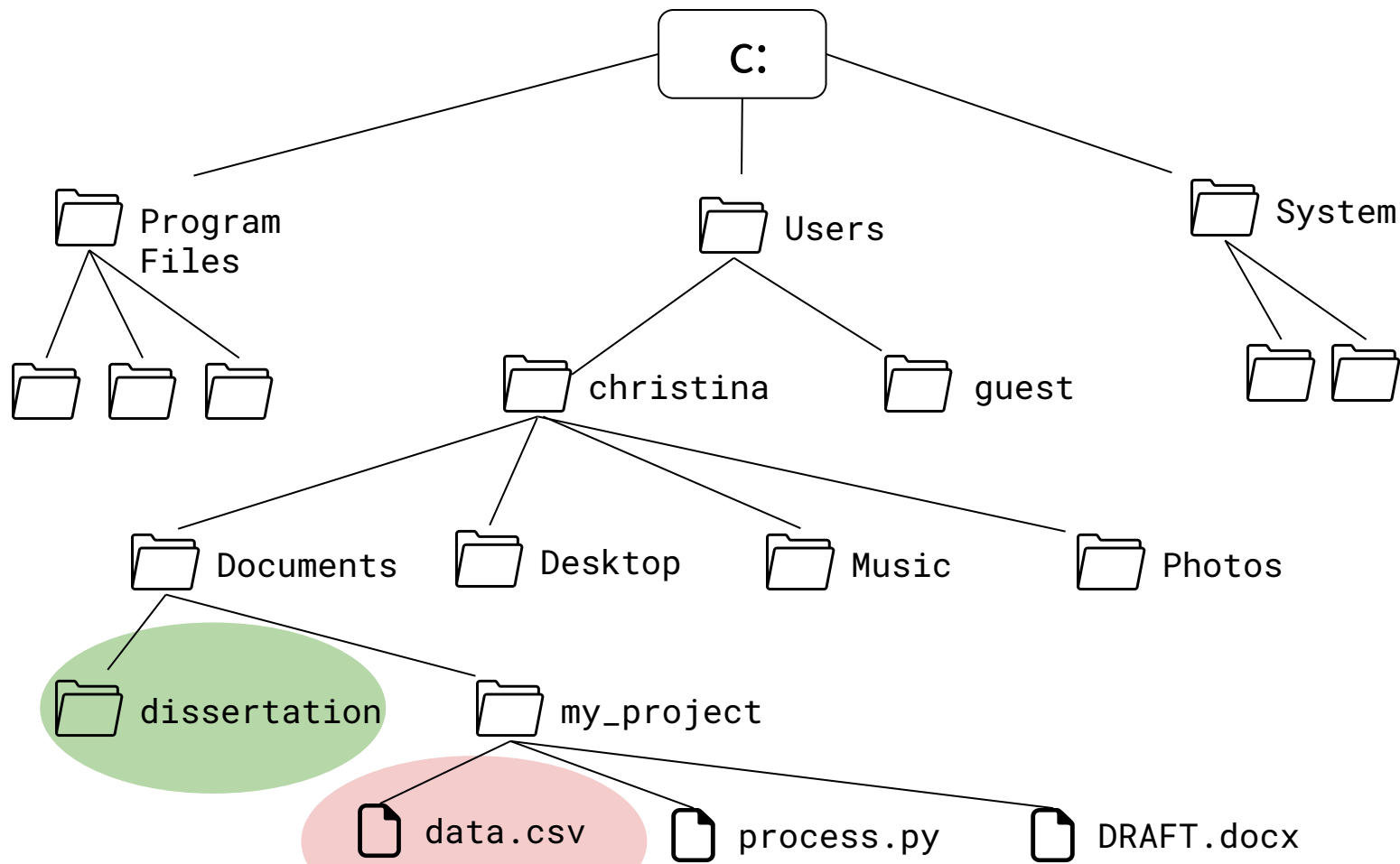
From Desktop: `../Documents/my_project/data.csv`



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

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

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?

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 [ResearchGate]

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 a useful reference, but you'll need a database connection to run it. See that repository for more details.

[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	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	...
	...
	...
14	...
	...
15	
16	...
	...
	...
17	
18	

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
---	---	---	---	---	---	---	---	----	----	----	----	-----	-----	-----	-----	----	-----	-----	-----	-----	----	----	-----	-----	-----	-----	----	----

3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	...
	...
	...
14	...
	...
15	
16	...
	...
	...
17	
18	

4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
---	---	---	---	---	---	----	----	----	----	-----	-----	-----	----	-----	-----	----	----	-----	-----	-----	----	----

5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
...	...
...	...
...	...
14	14
...	...
...	...
15	15
16	16
...	...
...	...
...	...
17	17
18	18

6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
...	...
...	...
...	...
14	14
...	...
...	...
15	15
16	16
...	...
...	...
...	...
17	17
18	18

7	7
8	8
9	9
10	10
11	11
12	12
13	13
...	...
...	...
...	...
14	14
...	...
...	...
15	15
16	16
...	...
...	...
...	...
17	17
18	18

8		
9		
10		
11		
12		
13		
	...	
	...	
	...	
14		
	...	
	...	
15		
16		
	...	
	...	
	...	
17		
18		

9
10
11
12
13
...
...
...
14
...
...
15
16
...
...
...
17
18

10			
11			
12			
13	***	***	***

14	***	***	
15			
16	***	***	***
17			
18			

11			
12			
13	***	***	***
14	***	***	
15			
16	***	***	***
17			
18			

12				
13	***	***	***	
14	***	***		
15				
16	***	***	***	
17				
18				

13	***	***	***
14	***	***	
15			
16	***	***	***
17			
18			

14
15
16
17
18

14
15
16
17
18

14
15
16
17
18

14
15
16
17
18

15
16
17
18

15
16
17
1815
16
...
...
...
17
1816
17
1817
1817
1817
1817
18

18

! "\$" \$??gg"??

[illegible][illegible]

?*?\n??6?^???ءI K?????U

X?q?_lp?Y???Ic?%J??M?e?v?))??C??/#Z?[[???mx_Ē??&t+9P掄??IGs??#?s?\$\$?Lz??/?=@??=k~???ls???T??-D??絕?d?Z;Av9??
ZwPT??k???T?O?ü?ý???p?al?V.??? t??x&t??+J??p?f2k)C??U?3?ävvd?X?????E?4G????[)Bz?}??PG?^?U??i?<?Hk??A?Q?4u?2R???
KAX??;???qN?????I?B-X?T?h
(?[?1E??+7+??2i?礪?φ?I1

0Z2G5IXre?q??Y[??S??n?.=)[?]r|?3?R??F9?X'?0???W??n?n~??x?yX?????r\F?q[?]??vK?]- "H?uZ??ca
0???gpA?mG`??a??x??`?X?^%*r?MM'c?k?Y???y??0?Pm*rk?P?lL]?{yUA???b?+?3???+kK?x??L???s?yGWe`??"?:??Q):???3[Fz???.!r??
?fkzBIvAPA>????X?e?h\U??p?S#&??~K?A? ??????V:{??b

???U?.?B=h?%&??dhGtX???\~'?';J?w?+?C?|?7????m? ??{P???vrBv??_DB1????_H
?r????h??z?t???;))??l?q[lYW??g]B?H<b?Z?9+]N?

???"P????F,?{?i?#?x??w??5???&Ui<zTd?E?:?B\$????{Z?/??

z q

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1																		
2	Department of State																	
3	Bureau of Population, Refugees, and Migration																	
4	Office of Admissions - Refugee Processing Center																	
5	Demographical Profile of Refugee Arrivals																	
6	Calendar Year																	
7	Afghanistan																	
8	as of 21-February-2017																	
9																		
10	Arrivals by Demographic																	
11																		
12	Report Start Date:	1-January-2002																
13	Report End Date:	21-February-2017																
14																		
15	Characteristic	CY 2002			CY 2003			CY 2004			CY 2005			CY 2006				
16		F	M	Total	F	M	Total	F	M	Total	F	M	Total	F	M			
17	Under 14	413	440	853	187	236	423	116	147	263	141	155	296	93	101			
18	Age 14 to 20	286	185	471	156	204	360	97	114	211	82	90	172	69	88			
19	Age 21 to 30	161	37	198	106	90	196	58	49	107	71	49	120	32	56			
20	Age 31 to 40	205	21	226	88	63	151	56	38	94	63	59	122	47	31			
21	Age 41 to 50	128	20	148	92	58	150	55	51	106	36	82	118	33	45			
22	Age 51 to 64	57	39	96	32	35	67	12	26	38	14	24	38	13	18			
23	Age 65 and Over	15	19	34	9	14	23	9	8	17	5	5	10	1	6			
24	Total	1,265	761	2,026	670	700	1,370	403	433	836	412	464	876	288	345			
25	Data prior to 2002 was migrated into WRAPS from a legacy system therefore we have more confidence in post-2002 data.																	
26																		
27																		
28																		

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

"whitespace"

A diagram consisting of two arrows. One arrow originates from the right side of the text '\n New Line' and points towards the 'w' in 'whitespace'. The other arrow originates from the right side of the text '\t Tab' and points towards the 'h' in 'whitespace'.

"This is line 1.
This 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"

"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) =

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

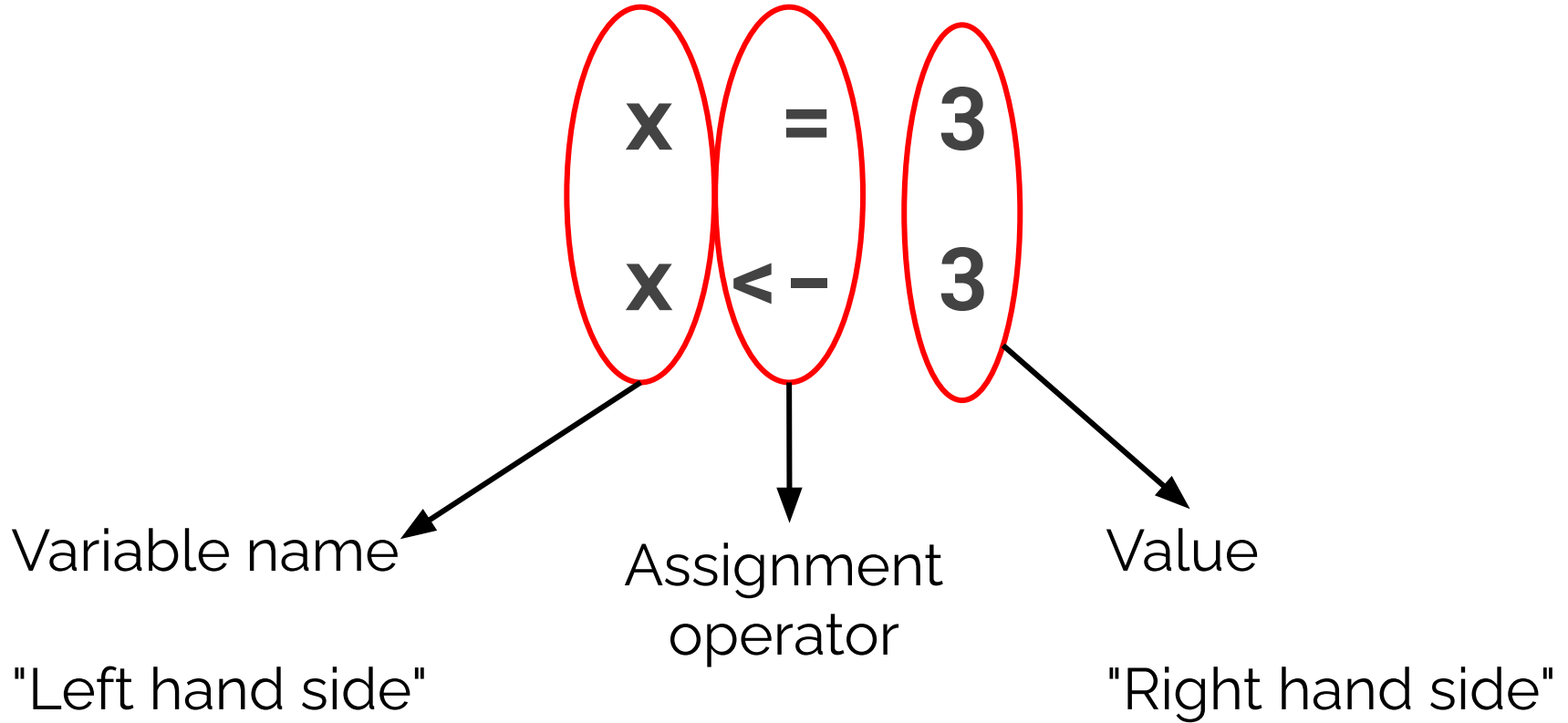


Variables

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



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

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]
```

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)

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

0 1 2 3 4 5 6 7

30

List Indexing (Python)

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

```
ages[1:4]
```

0 1 2 3 4 5 6 7

```
[30, 24, 24]
```

Python indexing is

exclusive of the end position

Vector Indexing (R)

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

1 2 3 4 5 6 7 8

```
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

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!"
else if age >= 16
    print "Learn to drive"
else
    print "Too young!"
```

Chained If Statements

<code>if age >= 18</code>	18, 19, 20, 21...
<code> print "Go Vote!"</code>	
<code>else if age >= 16</code>	16, 17
<code> print "Learn to drive"</code>	
<code>else</code>	15, 14, 13, 12,
<code> print "Too young!"</code>	11, 10...

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

```
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")
```

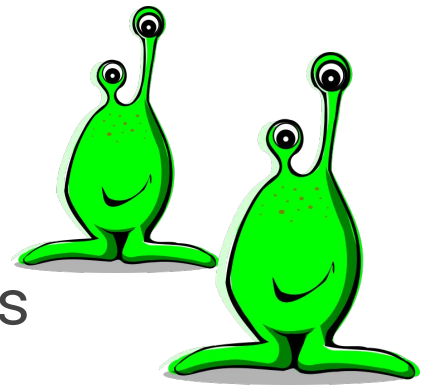
```
    print("Stay in school")
```

```
} else {
```

```
    print("Too young!")
```

```
}
```

Writing Pseudocode

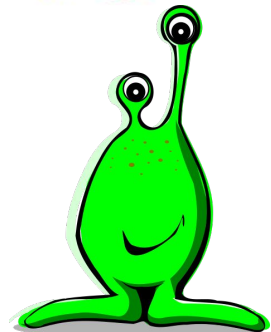
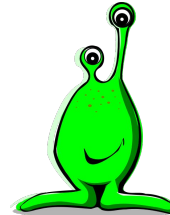
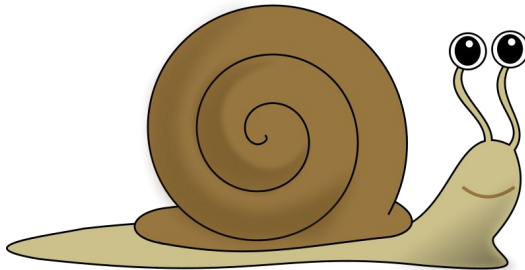


if color is green and has eye stalks

 send to Mars

else

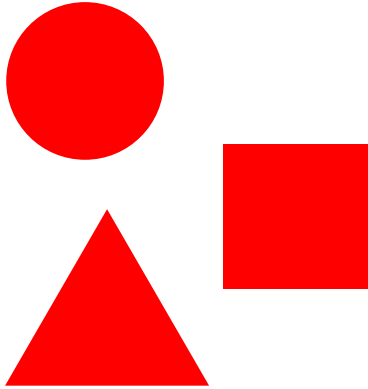
 send to Earth



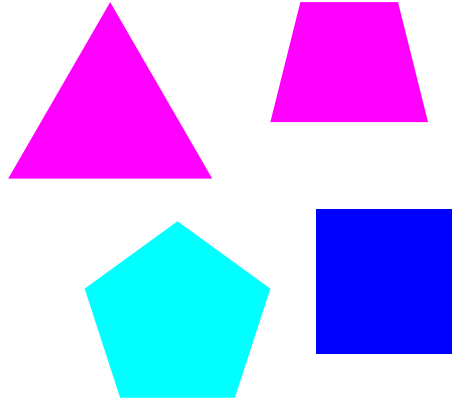
Exercise

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

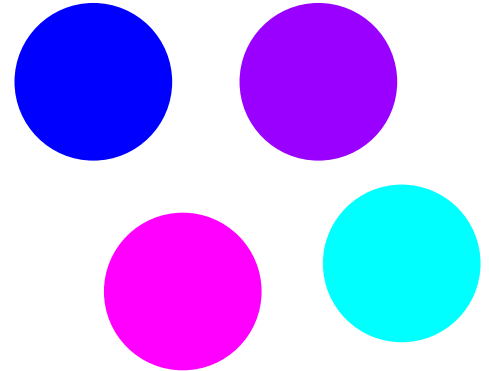
Group 1



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
```

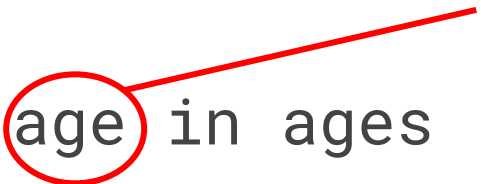
```
    do something
```

For Loop

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

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

Variable called age



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

for dog in row

bark(dog)



Ruff!

for dog in row

bark(dog)



Arf!

for dog in row

bark(dog)



for dog in row

bark(dog)

Hav!



for dog in row

bark(dog)

Voff!



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 1:8
```



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

```
    total = total + i
```

```
print total
```

For Loop

```
total = 0
```

```
for i in 1:8
```

```
    total = total + i
```

```
print total
```

i	total
	0
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36

For Loop

```
total = 0
```

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



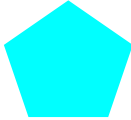

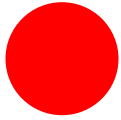


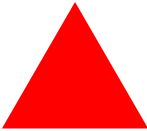
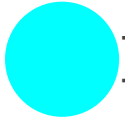
```
for i in ages
```

```
    total = total + i
```

```
print total
```

i	total
	0
42	42
30	72
24	96
24	120
27	147
35	182
39	221
22	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)



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)



Exercise for practicing loops at home:

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

Or search Google for "google doodle coding".

Functions



Functions

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

Functions

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

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!")
```

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://cran.r-project.org/manuals.html>

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

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: 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 = "", ...)
```

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

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

`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.** However...

Calling Functions: Argument order

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

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

`open("w", 'results.txt')` ❌

The rules about order work differently in Python than they do in R as you start adding more arguments.

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, 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

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	8666	20593360
Brazil	1999	37737	172006362
Brazil	2000	80488	174604898
China	1999	212258	1272915272
China	2000	213766	128042583

variables

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	8666	20593360
Brazil	1999	37737	172006362
Brazil	2000	80488	174604898
China	1999	212258	1272915272
China	2000	213766	128042583

observations

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	8666	20593360
Brazil	1999	37737	172006362
Brazil	2000	80488	174604898
China	1999	212258	1272915272
China	2000	213766	128042583

values

Rows: Observations

Person (or mouse, worm, etc.)

Country

Year

Run/trial of an experiment

Chemical

Sample

Columns: Variables

Measurements

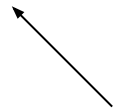
Grouping/identification variables:

- > trial/sample
- > condition
- > label for the observation (country name)

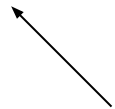
Each ID variable in its own column

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

R or Python?



Should you learn R or Python?

Should you learn R or Python?

Are you learning how to code for research?

Or to be a software developer?

Should you learn R or Python?

Are you learning how to code for research?

- Learn either R or Python - you can do pretty much everything now in either language

Or to be a software developer?

- Learn Python, Java, JavaScript, or C++

Should you learn R or Python?

Are you learning how to code for research?

- Learn either R or Python - you can do pretty much everything now in either language
- Which language is already used in your lab? These are the people you will go to when you have questions and need help.
- Which language is commonly used in your field or subfield? This is the code you will borrow when you are trying to apply published research methods to your own data.

Should you learn R or Python?

Are you learning how to code for research?

- Learn either R or Python - you can do pretty much everything now in either language
- Did you already try and fail to learn one language? Try the other, maybe you will like it better.

How to learn R or Python

Options for all types of learners:

- In-person/set schedule - NUIT Research Computing Services bootcamps, NICO Python course at NU
- Self-guided videos, books, interactive tutorials - NUIT Research Computing Services materials, or check out our lists of recommended resources:
 - <https://sites.northwestern.edu/researchcomputing/2020/03/31/online-learning-resources-python/>
 - <https://sites.northwestern.edu/researchcomputing/2020/03/20/online-learning-resources-r/>