REMINDER: help sessions

Mondays 3:50-5:00
this room (or WH 113)

except for today, which will be right after class

Homework 1

posted on Brightspace

due next Wed (Sep 7)

only worth 5 points (other assignments will be longer and worth more points) - mainly to check that everyone has Python installed properly

download from Brightspace

JupyterNotebooks.ipynb

VariablesAndTypes.ipynb

Modules.ipynb

LogicalTypes.ipynb

StringsRegularExpressions.ipynb

ListsTuples.ipynb

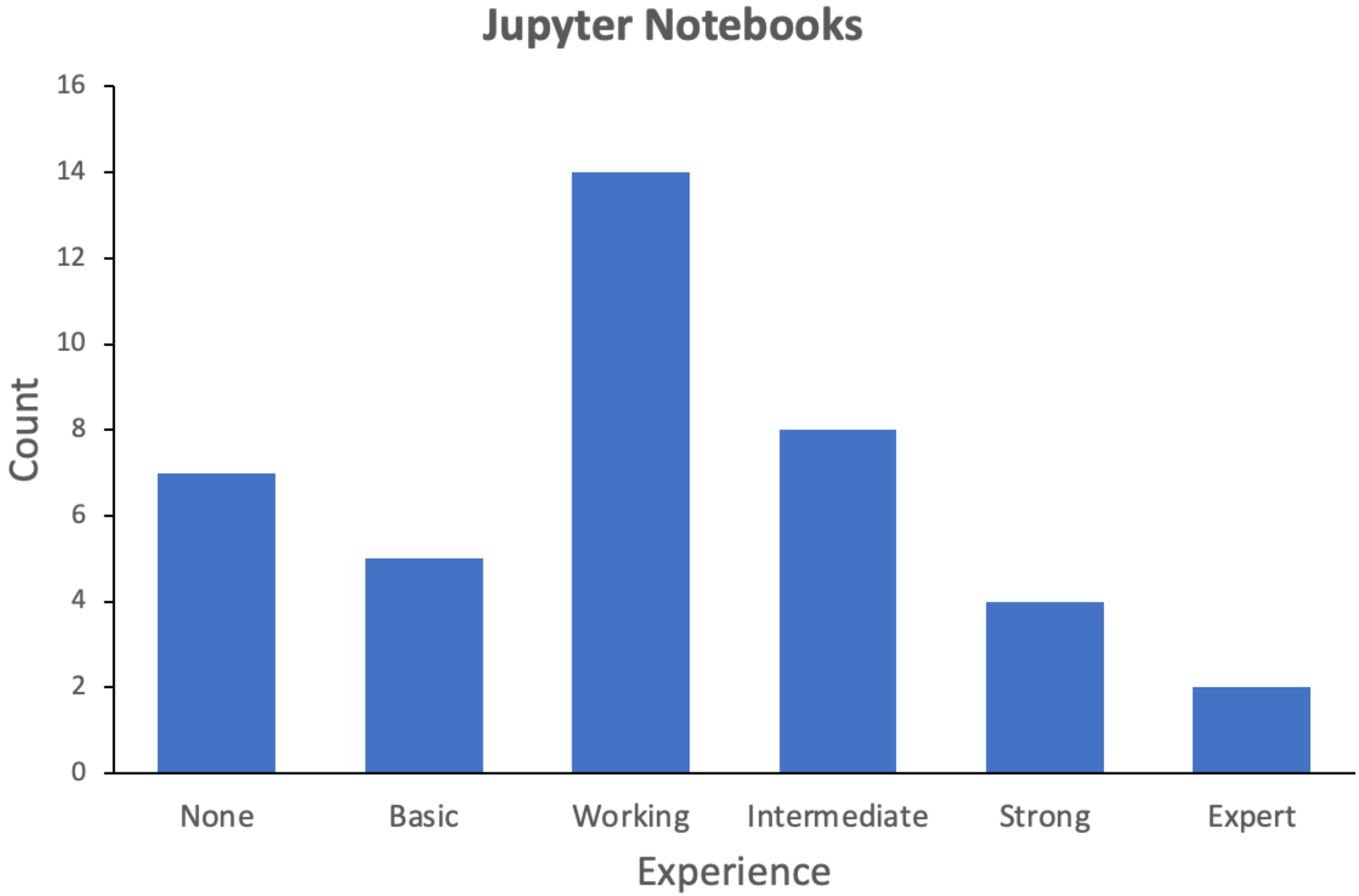
SetsAndDictionaries.ipynb

Jupyter Notebooks

we'll use PyCharm later - for now, we'll use Notebooks

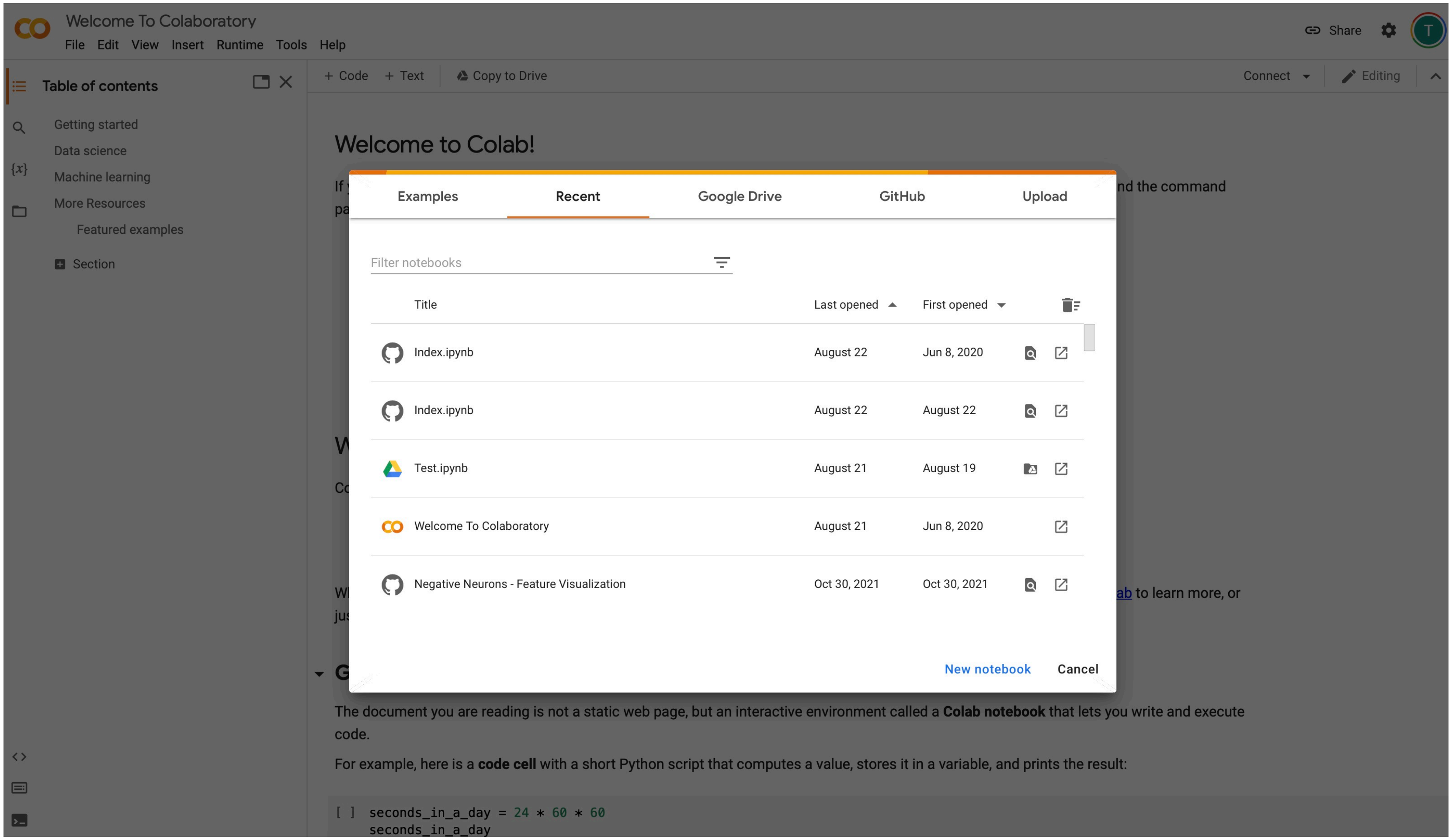
Jupyter Notebooks

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.


Jupyter Notebooks
Count
16
14
12
10
8
6
4
2
0
None
Basic
Working
Intermediate
Strong
Expert
Experience

Google Colab

https://colab.research.google.com



Welcome to Colab!

Title	Last opened	First opened
Index.ipynb	August 22	Jun 8, 2020
Index.ipynb	August 22	August 22
Test.ipynb	August 21	August 19
Welcome To Colaboratory	August 21	Jun 8, 2020
Negative Neurons - Feature Visualization	Oct 30, 2021	Oct 30, 2021

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
    seconds_in_a_day
```

can upload notebooks (.ipynb files), load notebooks from GitHub,
or create a new notebook, and obviously you can then download it

Google Colab

https://colab.research.google.com

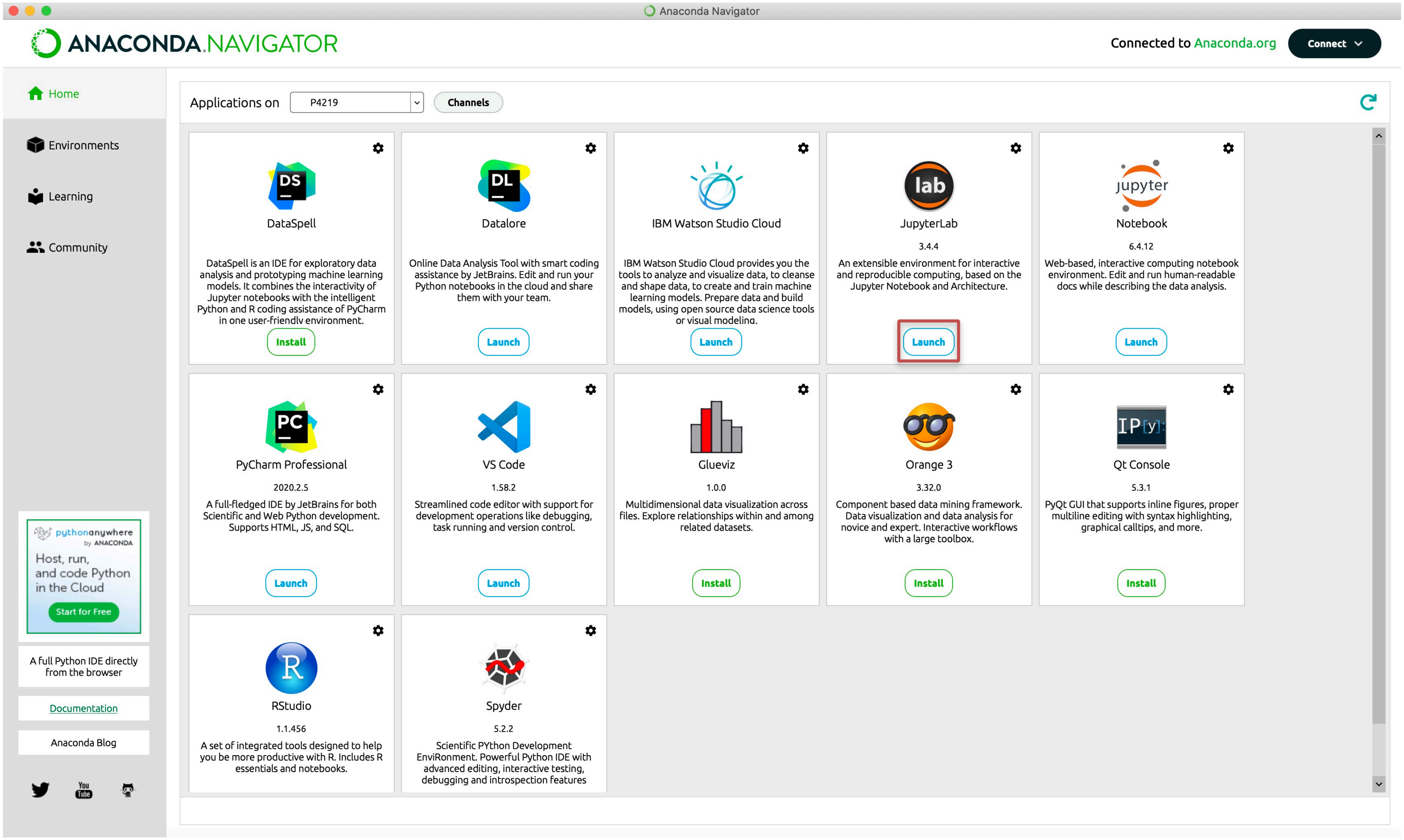
(part of) HOMEWORK 1:

upload `JupyterNotebooks.ipynb` to Google CoLab (on Brightspace) and run it

grab a screen shot (after it runs) and submit (with other parts)

we'll talk about how to run in a bit

launch JupyterLab (or Jupyter Notebook)



Applications on P4219

DataSpell

DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

Install

Datalore

Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.

Launch

IBM Watson Studio Cloud

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to cleanse and shape data, to create and train machine learning models. Prepare data and build models, using open source data science tools or visual modeling.

Launch

JupyterLab

3.4.4

An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Launch

Notebook

6.4.12

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Launch

PyCharm Professional

2020.2.5

A full-fledged IDE by JetBrains for both Scientific and Web Python development. Supports HTML, JS, and SQL.

Launch

VS Code

1.58.2

Streamlined code editor with support for development operations like debugging, task running and version control.

Launch

Glueviz

1.0.0

Multidimensional data visualization across files. Explore relationships within and among related datasets.

Install

Orange 3

3.32.0

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

Install

Qt Console

5.3.1

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Install

RStudio

1.1.456

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Spyder

5.2.2

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

A full Python IDE directly from the browser

Documentation

Anaconda Blog

you can also run from command line:

jupyter
Notebook

6.0.3

Windows 10 - from Command Prompt

```
C:\> jupyter notebook
```

MAC OSX - from Terminal

```
$ jupyter notebook
```

if you know how to activate environments
from the command line

you can also run from command line:

JupyterLab

2.1.5

Windows 10 - from Command Prompt

```
C:\> jupyter lab
```

MAC OSX - from Terminal

```
$ jupyter lab
```

if you know how to activate environments
from the command line

you can also run from command line:

JupyterLab

2.1.5

Windows 10 - from Command Prompt

```
C:\> jupyter lab
```

MAC OSX - from Terminal

```
$ jupyter lab
```

if you know how to activate environments
from the command line

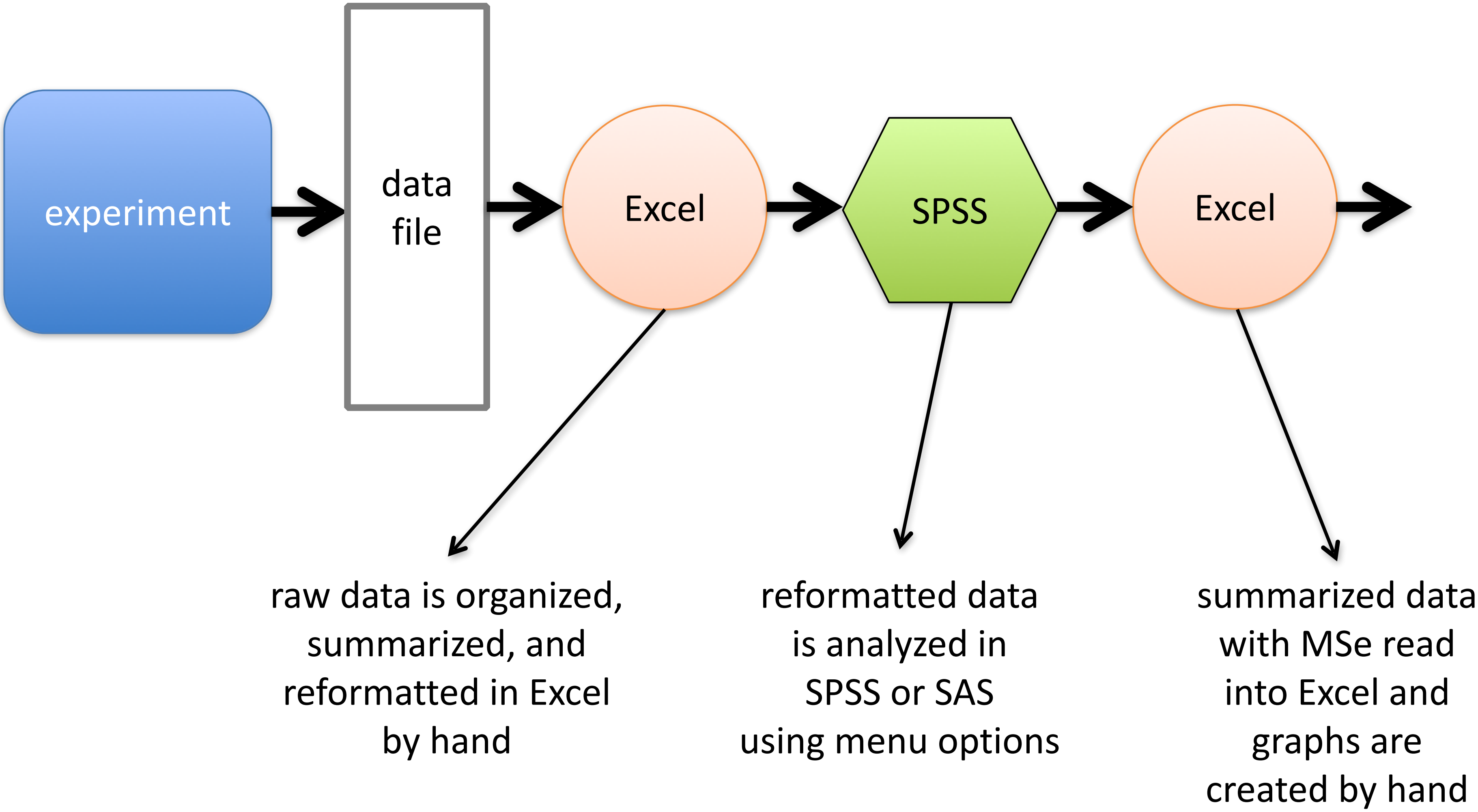
you can also run from the terminal
within an IDE like PyCharm

akin to lab digital notebooks

Jupyter Notebooks (are great) as Scripts

- Jupyter Notebooks are great for trying out snippets of Python code (as is the Python Console, which we will use within the PyCharm IDE)
- "Scripts" are a series of commands (though you can define functions within them). Scripts are not stand-alone programs nor are they modules/packages that can be used by other programs.
- Jupyter Notebooks are fantastic for fully documenting a data analysis / modeling pipeline. They are a computational equivalent of a paper lab notebook. Preferred over more ad hoc analysis pipelines commonly used.

common (old) data analysis pipeline



experiment

data file

Excel

SPSS

Excel

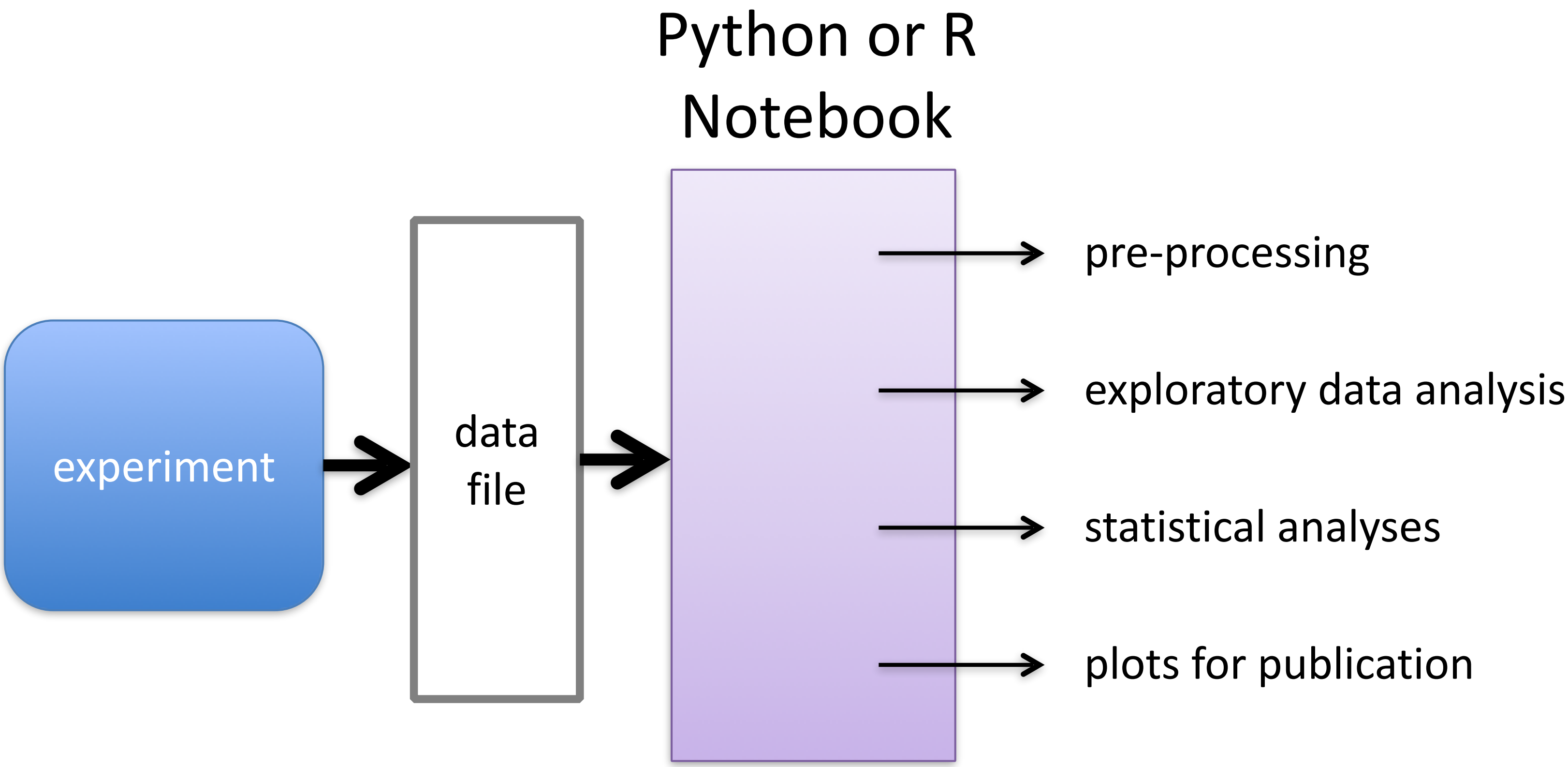
raw data is organized, summarized, and reformatted in Excel by hand

reformatted data is analyzed in SPSS or SAS using menu options

summarized data with MSe read into Excel and graphs are created by hand

maybe (hopefully) notes are kept
(paper or electronic, often separate from analyses)

Best Practice : Jupyter Notebooks



Python or R Notebook

experiment

data file

pre-processing

exploratory data analysis

statistical analyses

plots for publication

promises more reliable, robust, reproducible research

share data and Jupyter Notebook publicly (lab, GitHub, OSF)

using Jupyter

`JupyterNotebooks.ipynb`

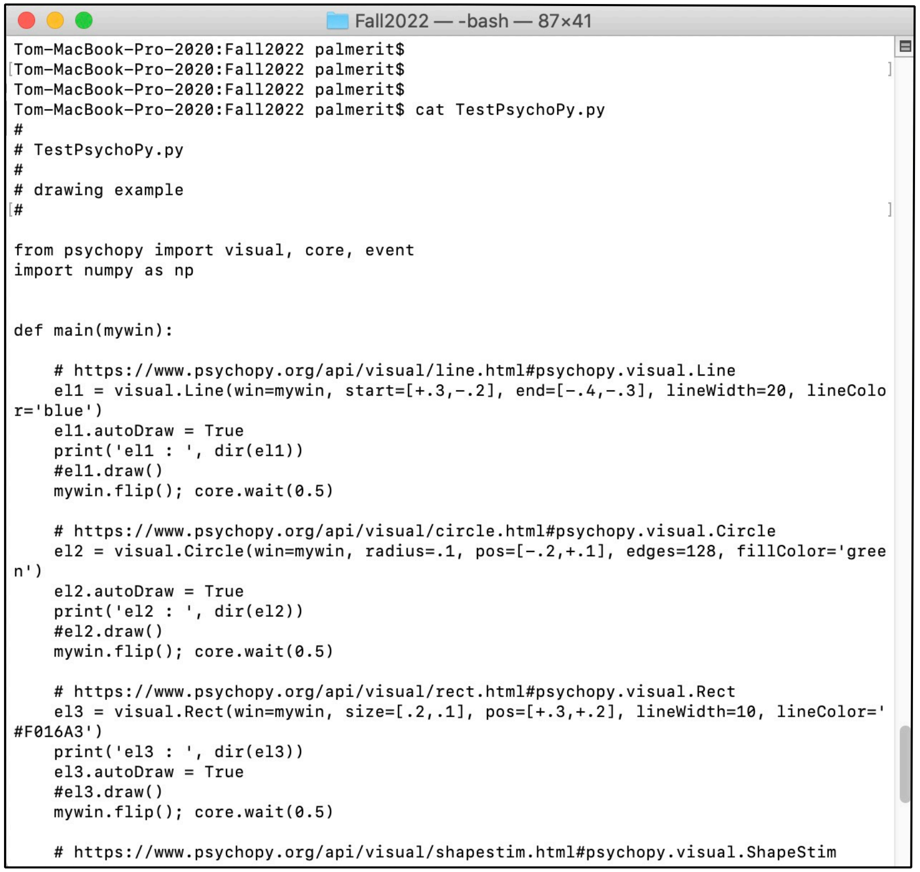
using JupyterLab (or Jupyter Notebook)

make sure you reset the kernel and clear all outputs and rerun your code before turning in

cells can be rerun in any order (note numbers next to cells), but we will run your code from top to bottom

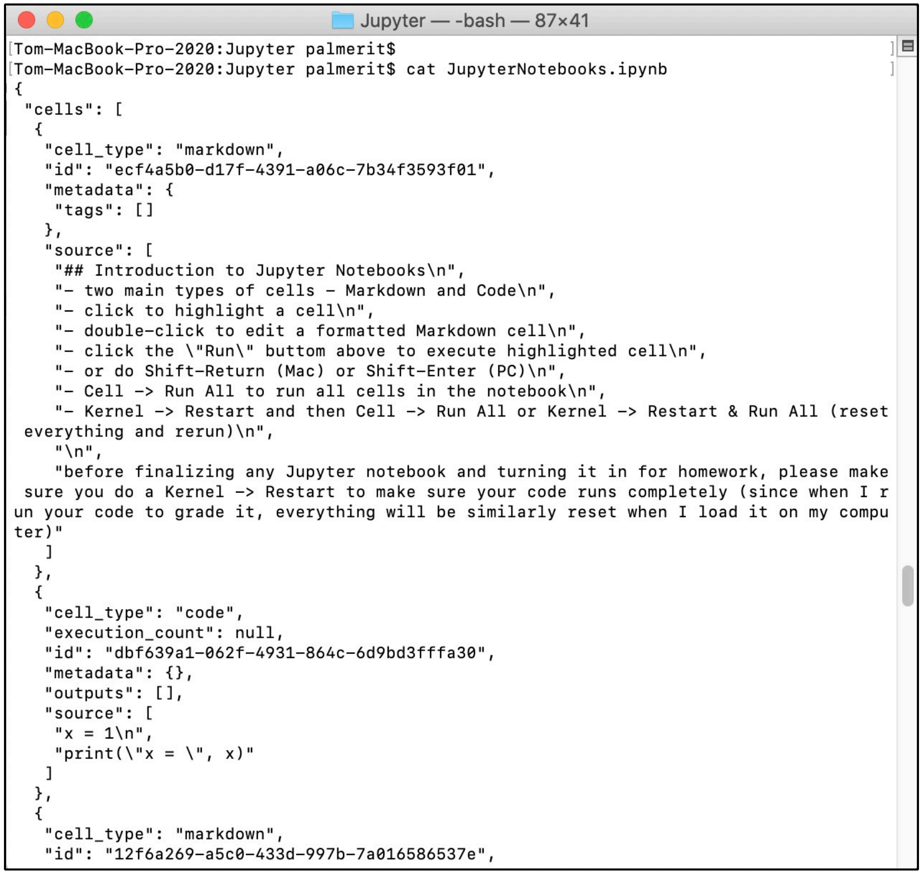
both .py and .ipynb files are text files

TestPsychoPy.py



```
Tom-MacBook-Pro-2020:Fall2022 palmerit$
Tom-MacBook-Pro-2020:Fall2022 palmerit$
Tom-MacBook-Pro-2020:Fall2022 palmerit$
Tom-MacBook-Pro-2020:Fall2022 palmerit$ cat TestPsychoPy.py
#
# TestPsychoPy.py
#
# drawing example
#

from psychopy import visual, core, event
import numpy as np


def main(mywin):

    # https://www.psychopy.org/api/visual/line.html#psychopy.visual.Line
    el1 = visual.Line(win=mywin, start=[+.3,-.2], end=[-.4,-.3], lineWidth=20, lineColo
r='blue')
    el1.autoDraw = True
    print('el1 : ', dir(el1))
    #el1.draw()
    mywin.flip(); core.wait(0.5)

    # https://www.psychopy.org/api/visual/circle.html#psychopy.visual.Circle
    el2 = visual.Circle(win=mywin, radius=.1, pos=[-.2,+.1], edges=128, fillColor='gree
n')
    el2.autoDraw = True
    print('el2 : ', dir(el2))
    #el2.draw()
    mywin.flip(); core.wait(0.5)

    # https://www.psychopy.org/api/visual/rect.html#psychopy.visual.Rect
    el3 = visual.Rect(win=mywin, size=[.2,.1], pos=[+.3,+.2], lineWidth=10, lineColor='
#F016A3')
    print('el3 : ', dir(el3))
    el3.autoDraw = True
    #el3.draw()
    mywin.flip(); core.wait(0.5)

    # https://www.psychopy.org/api/visual/shapestim.html#psychopy.visual.ShapeStim
```

JupyterNotebooks.ipynb



```
Tom-MacBook-Pro-2020:Jupyter palmerit$
Tom-MacBook-Pro-2020:Jupyter palmerit$ cat JupyterNotebooks.ipynb
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "ecf4a5b0-d17f-4391-a06c-7b34f3593f01",
   "metadata": {
    "tags": []
   },
   "source": [
    "## Introduction to Jupyter Notebooks\n",
    "- two main types of cells - Markdown and Code\n",
    "- click to highlight a cell\n",
    "- double-click to edit a formatted Markdown cell\n",
    "- click the \"Run\" buttom above to execute highlighted cell\n",
    "- or do Shift-Return (Mac) or Shift-Enter (PC)\n",
    "- Cell -> Run All to run all cells in the notebook\n",
    "- Kernel -> Restart and then Cell -> Run All or Kernel -> Restart & Run All (reset
 everything and rerun)\n",
    "\n",
    "before finalizing any Jupyter notebook and turning it in for homework, please make
 sure you do a Kernel -> Restart to make sure your code runs completely (since when I r
un your code to grade it, everything will be similarly reset when I load it on my compu
ter)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "dbf639a1-062f-4931-864c-6d9bd3fffa30",
   "metadata": {},
   "outputs": [],
   "source": [
    "x = 1\n",
    "print(\"x = \", x)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "12f6a269-a5c0-433d-997b-7a016586537e",
```

(JSON format)

Python Basics

Comments in Python

```
# this is a comment
x = 2   # this is a comment too
```

Use comments to:

- briefly describe what a section of code does
- reference particular formulas or calculations
- give credit to where borrowed code comes from
- describe variables the first time you assign them
- write comments for yourself
- write comments for others in your lab

Defining and Executing Cells in Python

```
#%%
```

Jupyter Notebooks are composed of Cells (self-containing portions of code) that can be independently executed (as shown earlier)

In Python code (.py files) you may also sometimes see this #%% notation to indicate the start of a Cell that some IDEs understand (PyCharm, Spyder) and can be executed like Cells in Jupyter Notebooks

Variables

`VariablesAndTypes.ipynb`

https://docs.python.org/3/tutorial/introduction.html

Variables

- A variable is the name of a place where data is stored in computer memory.
- Every language has rules for variable names. And not all the rules are the same.*
- Python variables
  - must start with a character (not a number)
  - can include lowercase, uppercase, number, and _
  - can start with a _ but usually signal special / reserved
  - are case-sensitive
  - can be any reasonable length

* e.g., **`my.var`** is an acceptable variable name in R, not in Python

Valid	Invalid
`x1`	`1x`
`x_fact`	`x!`
`sin`	`for`

some keywords (like `for` and `while`) are reserved

but Python lets you overwrite others (like `sin`)*

*can lead to difficult bugs to locate in Python

Best Practice

- use meaningful variable names
    - not `xxxy, xxxx, xyxyxy`
- but also use short variable names
    - `idx` vs. `index_counter`
- you can use letters from equations, if meaningful
    - `x`, `N`, `a`
- by convention, `i`, `j`, and `k` are often used to index loops, as counters

Types

(Classes)

`VariablesAndTypes.ipynb`

Variable Types in Python

- Every programming language defines different types for different kinds of data

  Scalar Types

  Numeric: `int`, `float`, `complex`

  Logical: `bool`

  Strings: `str`

  None Type: `None` (special)

  Compound Types

  list, tuple, dict, numpy array, pandas DataFrame

types define rules for
what kinds of operations
can be performed

Python is dynamically typed

- Some programming languages require you to declare the type of a variable explicitly. If you deviate from that type you get an error.

- Python lets you change types on the fly.

Type Casting in Python

- convert from one type to another

```
x = 1.1
y = int(x)

x = 1
y = float(x)

x = 125.411
y = str(x)
```

(class is "blueprint" for an object)

Everything in Python is an object

- objects are more complex than basic types in other programming languages (e.g., an `int` in C just tags the type and contains a value, that's it)
- attributes and methods associated with an object accessible using the . convention

`y` is an object

```
y = 1.
print(type(y))
print(y.is_integer())
```

`is_integer()` is a method associated with float object

Numeric Types

`VariablesAndTypes.ipynb`

Numeric Types

- int types

```
x = 1
y = 3
```

- float types

```
x = 1.
y = 3.4
```

Numeric Types

- int types

```
x = 1
y = 3
```

if you intend an integer number to be used as a counter or an index, it needs to be an `int` *type*

- float types

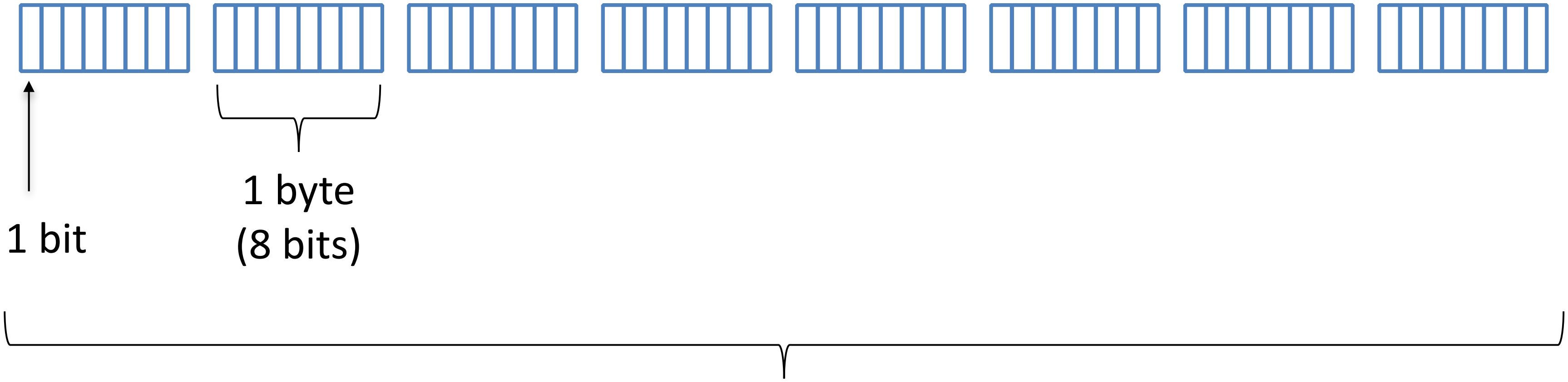
```
x = 1.
y = 3.4
```

if you intend an integer number to be used as a real number in calculations, add a decimal point to force it to be a `float` *type*

float Types

- every float in Python is “double-precision”

double-precision floating point



1 bit

1 byte
(8 bits)

8 bytes for one
double-precision number

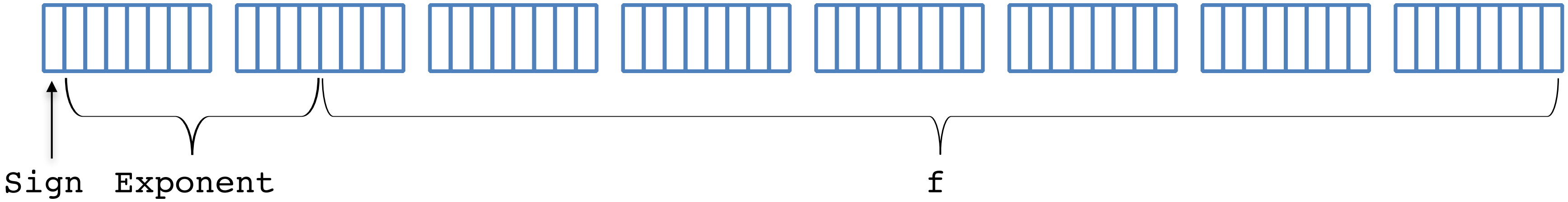
what’s important to understand is that there is
limited “real estate” to store significant digits of a number

so many floating point numbers are approximations

float Types

- every float in Python is "double-precision"

double-precision floating point



64 bits (8 bytes) per number in scientific notation

$$(\mathbf{Sign})\ 1.\mathbf{f}\ \texttt{x}\ 10^{\mathbf{Exponent}}$$

https://docs.python.org/3/tutorial/floatingpoint.html

https://www.wikihow.com/Convert-a-Number-from-Decimal-to-IEEE-754-Floating-Point-Representation

Significant Digits

```
print(1-.0000000001)
```

```
print(1-.0000000000000001)
```

```
print(1-.00000000000000001)
```

this is beyond the numeric resolution of Python

"... squeezing infinitely many real numbers into a finite number of bits requires an approximate representation ..." - Goldberg (1991)

What is 1000 x .01?

What is 1000 x .01?

```
print(1000 * .01)
```

What is 1000 x .01?

```
print(1000 * .01)

sum = 0
for i in range(0,1000):
    sum = sum+.01
print(sum)
```

review these slides and the
Jupyter notebook by Monday

Mathematical Operators

`VariablesAndTypes.ipynb`

Simple Mathematical Operations in Python

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
//	floor division
%	modulo division

https://jakevdp.github.io/WhirlwindTourOfPython/04-semantics-operators.html

What do you think these will give you?

```
2 ** 2+1

2+3 ** 2

2 * 3**2

4+2 / 1+2

-2 ** 2
```

What do you think these will give you?

```
(2 ** 2)+1
2+(3 ** 2)
2 * (3**2)
4+(2 / 1)+2
-(2 ** 2)
```

How Python interprets these

Order of Operations in Python

1) parentheses ( )
2) exponents **
3) positive (+) or negative (–) (not add or subtract)
4) multiplication (*) or division (/, //, %)
5) addition (+) or subtraction (–)
6) if equal OO, evaluated left-to-right

Best Practices

don’t rely on order of operations

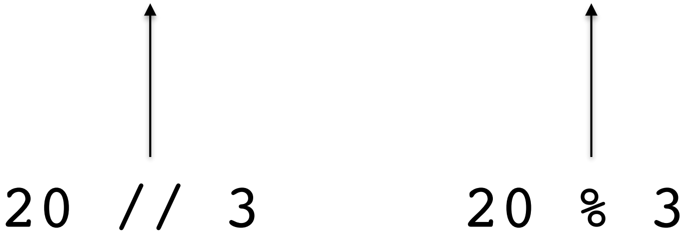
```
>> (-2)**2
>> -(2**2)
```

always use parentheses to ensure correctness and improve readability (even if they are unnecessary)

Modulo Division

`//` and `%` division

20 divided by 3 equals 6 remainder 2



```
20 // 3
```

```
20 % 3
```

like the way many of you first learned division in elementary school

Infinity in Python

```
x = float('inf')
print(x)
```

In Python, = is an assignment operator

```
x = 2
```

What does this do?

- if x does not exist
  - create it, reserve a place in memory for it, make it a float type (class), assign its value to 2
- if x does exist
  - see if the type needs to be changed, if so change it, assign its value to 2

~~`2 = x`~~

assignment in R:

```
x <- 2
```

compound assignment operators in Python

```
x = 2
x += 5    ⟷   x = x + 5
x -= 5    ⟷   x = x - 5
x *= 5    ⟷   x = x * 5
x /= 5    ⟷   x = x / 5
x %= 5    ⟷   x = x % 5
x //= 5   ⟷   x = x // 5
x **= 5   ⟷   x = x ** 5
```

review these slides and the Jupyter notebook by Monday

Assignment vs. Equivalence

`VariablesAndTypes.ipynb`

the “equals sign” =

- What can = mean in mathematics (outside Python)?
  - assert equivalence
    2 + 1 = 3
  - ask for a solution
    2 + 1 = ?
  - pose an algebraic problem
    2 + x = 3
  - ask a question regarding equivalence
    does 2 + (3-1) = 3?

- In Python, equals (=) means assignment

```
x = 3
```

assignment in R:
x <- 2

Sometimes we want to ask about equivalence

Does 2 + (3-1) = 3?

How do we do that in Python?

```
2 + (3-1) == 3
```

returns a boolean (`bool`) `True` or `False`