ST445 Managing and Visualizing Data

Workshop 1: Working with Jupyter Notebook and GitHub

2023/2024 Autumn Term

Today's plan

1. Git + GitHub
2. Anaconda + Jupyter
3. Data and Python

Before we start

- Make sure have a GitHub accout
- Make sure you have Anaconda installed
- Make sure you have Jupyter installed

Please ask if you are having trouble with any of the above.

Git + GitHub

GitHub

GitHub provides Internet hosting for software development and *version control* using *Git*.

- It is the largest development platform in the world
- Millions of developers and companies build, ship, and maintain their software on GitHub
    - Companies: Meta, Google, ...
    - Packages: NumPy, Pandas, NetworkX, ...

Version control

Version control is the practice of tracking and managing changes to computer programs, documents, or other collections of information.

- Some examples:
    - Book editions (e.g. different editions of `Python for Data analysis`)
    - Wikipedia page revision history
    - Software versions (e.g `Pandas` versions on official page or GitHub)

In this workshop, we focus on version control for software development.

Why version control?

Consider the following situations:

- Hard drive died, all stored work and documents are gone
- Accidentally deleted some important files
- Trying to "improve" programs or some files but realise the previous work is better

In these situations, you wish you will have some ways to:

- Get / revert back to previous "workable" version
- Figure out what has changed from the previous version

Why version control? (continue)

Consider the following situations when working on a project with other people:

- Different people working on the same document, resulted in several versions or conflicted copy of the same document
    - Often we need to merge the file manually which is time consuming and likely to make mistakes
- Someone made some changes to the document, but not sure why such changes were made, or who made the change

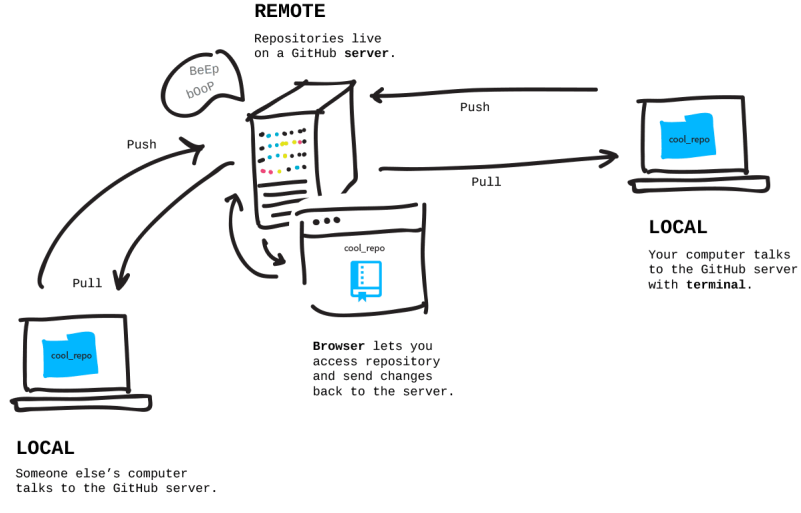
In these situations, you will wish you have some ways to:

- Compare and merge different versions / conflicts
- Have some information about who made the changes, when the changes were made and some notes about the changes

GitHub terminology

- Repository - a space for a project
- Clone - make a copy of the repository that lives on your computer
- Fork - create a remote copy of a repository that you can modify
- Branch - a parallel version of the repository
- Commit - save changes with a short description
- Push - send a Commit on your local branch to the remote repository
- Pull request - ask changes to be merged
- Merge - incorporate changes (then delete branch)

We will focus on how "clone", "commit", "push" and "pull" work in this workshop.



Interacting with GitHub

- GitHub desktop
- In the terminal, using git

Exercise: fork and clone the ST445 repo, which can be found at this link.

Anaconda + Jupyter

Anaconda

- Open-source cross-platform distribution of the Python programming language
    - *conda* – package management system
    - *pandas*, *scikit-learn*, *nltk*, etc. – packages for data science
    - Anaconda Navigator – graphical user interface

Jupyter (notebook / lab)

A web-based interactive computational environment for creating notebook documents.

Exercise: find the ST445 repo you have cloned and open this notebook on your machine.

Add equations in Jupyter notebook

https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/Typesetting%20Equations.html

Latex math and equations

https://www.latex-tutorial.com/tutorials/amsmath/

Exercise: print the four equations shown on the board, be sure to allign them in a proper way

Data and Python

How data is represented in a computer

- Bit is a basic unit of information in a computer
- In a computer, data is represented using a sequence of bits
    - Examples we have seen in the lecture:
        - Unsigned integers
        - Integers
        - Real numbers (float)
        - Characters

How data is represented in a computer (continue)

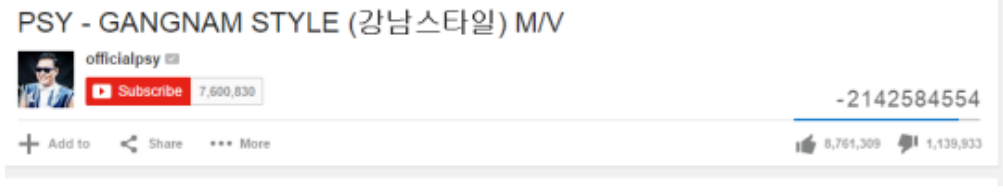
Some details:

- Integers:
    - With 64 bits we can represent integers $[-2^{64-1}, 2^{64-1} - 1]$
- Real numbers:
    - Given finite number of bits, many real numbers in the computer are only *approximated*
- Characters:
    - Character encoding: map characters to numbers
    - Example: UTF-8 with Unicode, use 1-4 bytes to represent a character

Why we care about how data is represented in a computer?

We have seen some problems when:

- Representing integers out of the range that the given number of bits can represent:



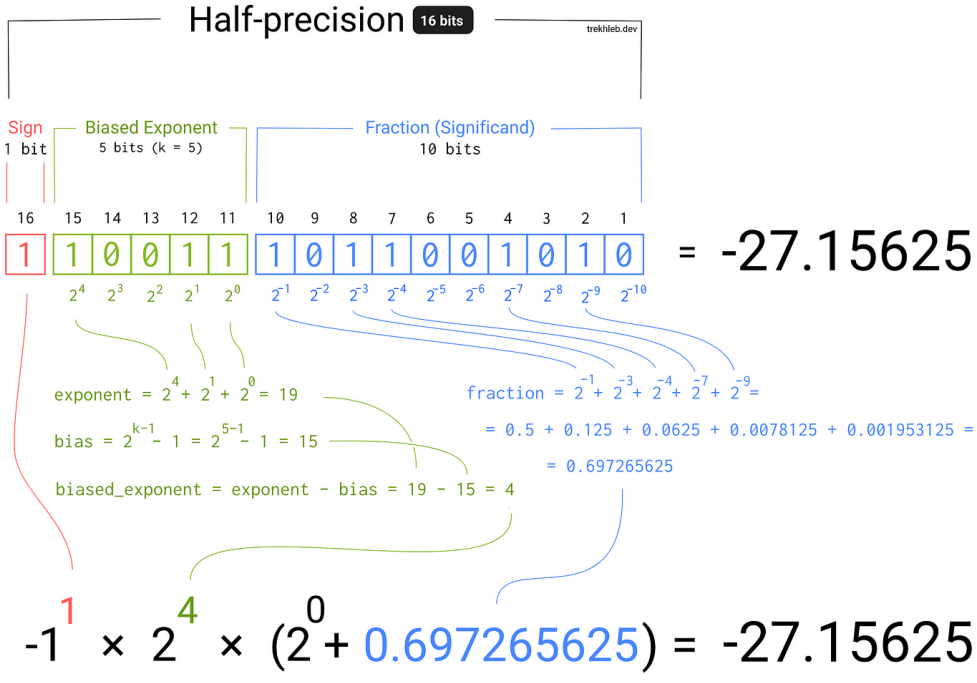
- Comparing `float`s. Example:

In [7]:
```
6.14 - 3.14 == 3
```

Out[7]:
```
False
```

In [9]:
```
float(6.14 - 3.14)
```

Out[9]:
```
2.9999999999999996
```



- Using different character encoding when encoding and decoding the textual data. Example:

Integers in Python

In Python, integers have "unlimited" precision.

- Python supports arbitrary precision integers (or "bignums") for the built-in integer type which can work with arbitrarily large integers
    - It uses variable-length arrays of digits
- The amount of available address space forms a practical limit
    - A computer has only a finite amount of storage, so they too can only represent a finite subset of the mathematical integers

Integer in Python: demo

The largest integer that 64 bits can represent is $2^{64-1} - 1 = 9,223,372,036,854,775,807$:

In [3]:
```
2**(64-1)-1
```

Out[3]:
```
9223372036854775807
```

But we can have much larger integers in Python. Example:

In [4]:
```
x = 100**500
x
```

Out[4]:
```
100000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000
```

```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000
```

Integer in Python: with `NumPy`

In `NumPy` (we will learn more about it next week), integer types use finite number of bits.

For example, `np.unit64` for unsigned integer with 64 bits:

In [21]:
```
import numpy as np
arr = np.array([2**32-1], dtype = np.uint32)
arr
```

Out[21]:
```
array([4294967295], dtype=uint32)
```

In [22]:
```
2**(32-1)
```

Out[22]:
```
2147483648
```

And *integer overflow* can happens:

In [23]:
```
arr + 1
```

Out[23]:
```
array([0], dtype=uint32)
```

(Can you explain why it is `0`?)

Why variable number of bits in Python? Why fixed number of bits in NumPy?

- variable number of bits: Eliminate the possibility of incorrect results due to simple overflow
    - Serious consequence of integer overflow:
        - Ariane 5 rocket by European Space Agency exploded 37s after its lift-off on 4th June 1996 - coined as the bug that cost £240m
- Advantages of using fixed number of bits:
    - Faster, better performance

Some Python data types

Lists are used to store multiple items, not necessarily of the same type, in a single variable.

- Ordered
- can contain arbitrary objects
- Elements can be accessed by index
- Allow duplicates
- Can be nested
- Mutable
- Dynamic

Tuples are used to store multiple items in a single variable.

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

In [3]:
```
myList = ["cat", "dog", "mouse"]
```

Some Python exercises

Exercise: try to debug the following code

In [1]:
```
#from IPython.core.debugger import set_trace
ShoppingList=['apples','pears','bananas']
#set_trace()
ShoppingList.Sort()
print(ShoppingList)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[1], line 4
      2 ShoppingList=['apples','pears','bananas']
      3 #set_trace()
----> 4 ShoppingList.Sort()
      5 print(ShoppingList)

AttributeError: 'list' object has no attribute 'Sort'
```