# Lab Assignment 1: How to Get Yourself Unstuck - Rachel Holman

## DS 6001: Practice and Application of Data Science

### Problem 0

*Import the following libraries:*

In [1]:
```
import numpy as np
import pandas as pd
import os
import math
```

### Problem 1

*Refer to the following Stack Overflow post: https://stackoverflow.com/questions/11346283/renaming-columns-in-pandas/46912050. How many unique different solutions were proposed?*

*Given the number of proposed solutions on this Stack Overflow page, what's the problem with developing a habit of using Google and Stack Overflow as your first source for seeking help? (2 points)*

There are **14** unique different solutions that were proposed:

- df.rename()
- df.set_axis()
- .columns attribute
- df.columns.str.replace()
- pd.concat()
- dtype
- transpose
- pd.DataFrame()
- comluns=dict(zip()
- col.strip("")
- str.slice(1)
- re.split(matchPattern, i)
- pd.Series()
- columns.str.replace()
- df.columns.str.lstrpi("")

The problem with developing a habit of using Google and Stack Overflow as the first source when seeking help is that a lot of the solutions you find may be much higher-level options than the coder is ready for. Additionally, sources like this show various different ways to use the same functions which can be confusing. For example, this post has 14 unique solutions but 35 replies so there is a lot of duplicates and different ways to apply the same solution.

## Problem 2

*Write code to display the docstrings for both the* `numpy` *and* `math` `log()` *functions.*

*Read the docstrings and explain, in words in your lab report, whether it is possible to use each function to calculate* $log_3(7)$ *or not. Why did you come to this conclusion?*

*If possible, use one or both functions to calculate* $log_3(7)$ *and display the output. (2 points)*

In [2]:
```
?np.log
```

```
Call signature:  np.log(*args, **kwargs)
Type:            ufunc
String form:     <ufunc 'log'>
File:            ~/anaconda3/lib/python3.10/site-packages/numpy/__init__.py
Docstring:
log(x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None,
subok=True[, signature, extobj])

Natural logarithm, element-wise.

The natural logarithm `log` is the inverse of the exponential function,
so that `log(exp(x)) = x`. The natural logarithm is logarithm in base
`e`.

Parameters
----------
x : array_like
    Input value.
out : ndarray, None, or tuple of ndarray and None, optional
    A location into which the result is stored. If provided, it must have
    a shape that the inputs broadcast to. If not provided or None,
    a freshly-allocated array is returned. A tuple (possible only as a
    keyword argument) must have length equal to the number of outputs.
where : array_like, optional
    This condition is broadcast over the input. At locations where the
    condition is True, the `out` array will be set to the ufunc result.
    Elsewhere, the `out` array will retain its original value.
    Note that if an uninitialized `out` array is created via the default
    ``out=None``, locations within it where the condition is False will
    remain uninitialized.
**kwargs
    For other keyword-only arguments, see the
    :ref:`ufunc docs <ufuncs.kwargs>`.

Returns
-------
y : ndarray
    The natural logarithm of `x`, element-wise.
    This is a scalar if `x` is a scalar.

See Also
--------
log10, log2, log1p, emath.log

Notes
-----
Logarithm is a multivalued function: for each `x` there is an infinite
number of `z` such that `exp(z) = x`. The convention is to return the
`z` whose imaginary part lies in `[-pi, pi]`.

For real-valued input data types, `log` always returns real output. For
each value that cannot be expressed as a real number or infinity, it
yields ``nan`` and sets the `invalid` floating point error flag.

For complex-valued input, `log` is a complex analytical function that
has a branch cut `[-inf, 0]` and is continuous from above on it. `log`
handles the floating-point negative zero as an infinitesimal negative
number, conforming to the C99 standard.

References
```

```
----------
.. [1] M. Abramowitz and I.A. Stegun, "Handbook of Mathematical Functions",
       10th printing, 1964, pp. 67.
       https://personal.math.ubc.ca/~cbm/aands/page_67.htm
.. [2] Wikipedia, "Logarithm". https://en.wikipedia.org/wiki/Logarithm

Examples
--------
>>> np.log([1, np.e, np.e**2, 0])
array([  0.,   1.,   2., -Inf])
Class docstring:
Functions that operate element by element on whole arrays.

To see the documentation for a specific ufunc, use `info`.  For
example, ``np.info(np.sin)``.  Because ufuncs are written in C
(for speed) and linked into Python with NumPy's ufunc facility,
Python's help() function finds this page whenever help() is called
on a ufunc.

A detailed explanation of ufuncs can be found in the docs for :ref:`ufuncs`.

**Calling ufuncs:** ``op(*x[, out], where=True, **kwargs)``

Apply `op` to the arguments `*x` elementwise, broadcasting the arguments.

The broadcasting rules are:

* Dimensions of length 1 may be prepended to either array.
* Arrays may be repeated along dimensions of length 1.

Parameters
----------
*x : array_like
    Input arrays.
out : ndarray, None, or tuple of ndarray and None, optional
    Alternate array object(s) in which to put the result; if provided, it
    must have a shape that the inputs broadcast to. A tuple of arrays
    (possible only as a keyword argument) must have length equal to the
    number of outputs; use None for uninitialized outputs to be
    allocated by the ufunc.
where : array_like, optional
    This condition is broadcast over the input. At locations where the
    condition is True, the `out` array will be set to the ufunc result.
    Elsewhere, the `out` array will retain its original value.
    Note that if an uninitialized `out` array is created via the default
    ``out=None``, locations within it where the condition is False will
    remain uninitialized.
**kwargs
    For other keyword-only arguments, see the :ref:`ufunc docs <ufuncs.kwargs>
`.

Returns
-------
r : ndarray or tuple of ndarray
    `r` will have the shape that the arrays in `x` broadcast to; if `out` is
    provided, it will be returned. If not, `r` will be allocated and
    may contain uninitialized values. If the function has more than one
    output, then the result will be a tuple of arrays.
```

In [3]:
```
?math.log
```

```
Docstring:
log(x, [base=math.e])
Return the logarithm of x to the given base.

If the base not specified, returns the natural logarithm (base e) of x.
Type:      builtin_function_or_method
```

The log function with **numpy** computes the natural logarithm, log (base e), so there is no way for the user to specify a different base (like 3 in this example). for this reason, there is no way to use numpy.log() to solve $log_3 7$ without using a change-of-base-formula.

The log function in **math** does allow you to specify both a value and a base number, so computing $log_3 7$ is as easy as:

```
math.log(7, 3)
```

In [4]:
```
math.log(7, 3)
```

Out[4]:
```
1.7712437491614221
```

## Problem 3

*Open a console window and place it next to your notebook in Jupyter labs. Load the kernel from the notebook into the console, then call up the docstring for the* `pd.DataFrame` *function. Take a screenshot and include it in your lab report. (2 points)*

## Problem 4

*Search through the questions on Stack Overflow tagged as Python questions: https://stackoverflow.com/questions/tagged/python. Find a question in which an answerer exhibits passive toxic behavior as defined in this module's notebook. Provide a link, and describe what specific behavior leads you to identify this answer as toxic. (2 points)*

This Stack Overflow post is a clear example of passive toxic behavior

Not only have many answerers and Stack Overflow users downvoted this question 22 times, but there are also very negative and shaming comments such as:

> "What else would you expect to happen? If you do something twice, it gets done twice."

and

> ""this function did exactly what I told it to do. why did it do that?" basically... "

## Problem 5

*Search through the questions on Stack Overflow tagged as Python questions: https://stackoverflow.com/questions/tagged/python. Find a question in which a questioner self-sabotages by asking the question in a way that the community does not appreciate. Provide a link, and describe what the questioner did specifically to annoy the community of answerers. (2 points)*

This Stack Overflow post is a clear case of the questioner self-sabotaging.

This user asked a question that had already been answered in a different Stack Overflow post, which annoyed the answering community so the user recieved 45 downvotes.

## Problem 6

These days there are so many Marvel superheros, but only six superheros count as original Avengers: Hulk, Captain America, Iron Man, Black Widow, Hawkeye, and Thor. I wrote a function, `is_avenger()`, that takes a string as an input. The function looks to see if this string is the name of one of the original six Avengers. If so, it prints that the string is an original Avenger, and if not, it prints that the string is not an original Avenger. Here's the code for the function:

In [5]:
```
def is_avenger(name):
    if name in ("Hulk", "Captain America", "Iron Man", "Black Widow", "Hawkeye'
        print(name  + "'s an original Avenger!")
    else:
        print(name + " is NOT an original Avenger.")
```

To test whether this function is working, I pass the names of some original Avengers to the function:

In [6]:
```
is_avenger("Black Widow")
```

```
Black Widow's an original Avenger!
```

In [7]:
```
is_avenger("Iron Man")
```

```
Iron Man's an original Avenger!
```

In [8]:
```
is_avenger("Hulk")
```

```
Hulk's an original Avenger!
```

Looks good! But next, I pass some other strings to the function

In [9]:
```
is_avenger("Spiderman")
```

```
Spiderman is NOT an original Avenger.
```

In [10]:
```
is_avenger("Beyonce")
```

```
Beyonce is NOT an original Avenger.
```

Beyonce is a hero, but she was too busy going on tour to be in the Avengers movie. Also, Spiderman definitely was NOT an original Avenger. It turns out that this function will display that any string we write here is an original Avenger, which is incorrect. To fix this function, let's turn to Stack Overflow.

## Part a

*In your lab report, write the link to the Stack Overflow page used to resolve the problem, and the search terms you entered into Google to find this page.*

*Then apply the solution on this Stack Overflow page to fix the* `is_avenger()` *function, and test the function to confirm that it works as we expect. (2 points)*

After Google searching:

> python if name== "x" or "y"

I found this Stack Overflow post which very clearly explains the problem and offers 2 different solutions.

By changing

```
if name=="Hulk" or "Captain America" or "Iron Man" or "Black
Widow" or "Hawkeye" or "Thor":
```

into

```
if name in ("Hulk", "Captain America", "Iron Man", "Black
Widow", "Hawkeye", "Thor"):
```

the function now works as we expect.

## Part b

*Suppose that no Stack Overflow posts yet existed to help us solve this problem. It would be time to consider writing a post ourselves. In your lab report, write a good title for this post. Do NOT copy the title to the posts you found for part a. (Hint: for details on how to write a good title see the slides or https://stackoverflow.com/help/how-to-ask) (3 points)*

> How to create "if" statement in Python with multiple conditions?

## Part c

*Write a minimal working example for this problem. (3 points)*

In [11]:
```
value = "d"

if value == "a" or "b" or "c":
    print("Value is a, b, or c.")
else:
    print("Value is not a, b, or c.")
```

```
Value is a, b, or c.
```

## Problem 7

*Sign on to the PySlackers slack page and send me a private message in which you tell me which three channels on that Slack workspace look most interesting to you. (2 points)*

Done!