functions_lambdas_help

February 1, 2022

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
```

0.1 Functions, lambda functions, and reading help documents

We are going to introduce a few more python concepts. If you've been working through the NHANES example notebooks, you will have seen these in use already. There is a lot to say about these new concepts, but we will only be giving a brief introduction to each. For more information, follow the links provided or do your own search for the many great resources available on the web.

0.1.1 Functions

If you use a snippet of code multiple times, it is best practice to put that code into a function instead of copying and pasting it. For example, if you wanted several of the same plots with different data, you could create a function that returns that style of plot for arbitrary (though with correct dimension and type) data.

In Python, indentation is very important. If done incorrectly, your code will not run and instead will give an error. When defining a function, all code after the ':' must be indented properly. The indentation conveys the scope of the code. Some further explanation.

```
def function_name(arguments):
    """
    Header comment: brief description of what this function does

    Args:
        obj: input for this function
    Returns:
        out: the output of this function
    """

    some code

    return out
```

Exactly how to structure the header comments is up to you if you work alone, or will likely be specified if working for an established company.

Function names should start with a lower case letter (they cannot start with a number), and can be in camelCase or snake_case.

If your function returns a variable, you use 'return' to specify that variable. A function doesn't always have to return something though. For example, you could have a function that creates a plot and then saves it in the current directory.

```
In [2]: def sum_x_y(x, y): # don't need comments if immediately clear what the function does
            out = x + y
            return out

        sum_x_y(4, 6)
```

```
Out[2]: 10
```

```
In [3]: def get_max(x):
            current_max = x[0]
            for i in x[1:]:
                if i > current_max:
                    current_max = i
            return current_max
```

```
In [4]: get_max(np.random.choice(400, 100))
        # np.random.choice(400, 100) will randomly choose 100 integers between 0 and 400
```

```
Out[4]: 387
```

There is a lot more to be said about functions that we don't have time to cover in this course, so I leave you with examples of common gotchas that you may run into.

0.1.2 lambda functions

There are also know as anonymous functions because they are unnamed. This function can have any number of arguments but only one expression. Lambda functions, unlike defined functions, always return a variable. The format of a lambda function is

```
lambda arguments: expression
```

They can look similar to a mathematical expression for evauating a function.
For example:

```
(lambda x: x**2)(3)
```

Is the same as mathmatically writing
$f(x) = x^2$ an then evauluating the function f at $x = 3$,
$f(3) = 9$

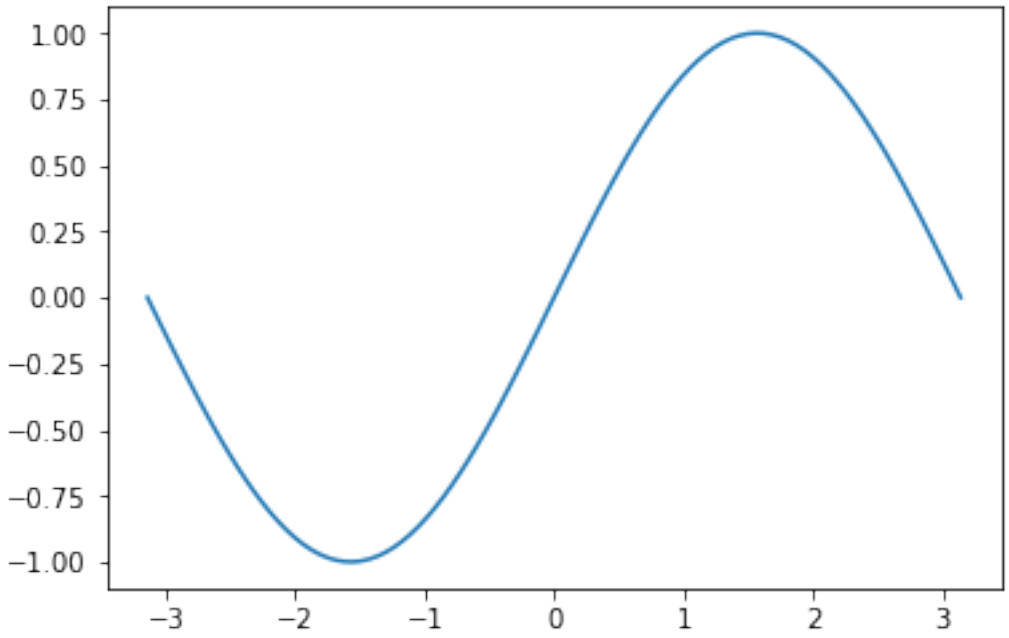
```
In [5]: (lambda x: x**2)(3)
```

```
Out[5]: 9
```

Another way to use a lambda function is to store it in a variable like in the example below.

```
In [6]: f = lambda x: np.sin(x)
        x = np.linspace(-np.pi, np.pi, 100)
        y = [f(i) for i in x]
        plt.plot(x, y)
        plt.show
        # we could have made this several ways, can you think of another?
```

```
Out[6]: <function matplotlib.pyplot.show(*args, **kw)>
```



You shouldn't come across many (if any) cases where you would have to use a lambda function, but we present them briefly here so that you can regonize them in the wild.

0.1.3 Reading help documentation

A key skill in being a successful programmer is being able to read the documentation for a function and understand what that functions does and what the arguments are.

To get the documentation, use the help function. First, let's call the help function on help, to see what is does:

```
In [7]: help(help)
```

```
Help on _Helper in module _sitebuiltins object:
```

```
class _Helper(builtins.object)
 |  Define the builtin 'help'.
 |
 |  This is a wrapper around pydoc.help that provides a helpful message
 |  when 'help' is typed at the Python interactive prompt.
 |
 |  Calling help() at the Python prompt starts an interactive help session.
 |  Calling help(thing) prints help for the python object 'thing'.
 |
 |  Methods defined here:
 |
 |  __call__(self, *args, **kwds)
 |      Call self as a function.
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

We can see that calling help(thing) will print the documentation for 'thing'. Generally, this documentation will first list the function with its arguments (also called parameters), showing what the default arguments are. Then, it will list these arguments (parameters) and specify what they are and their type. Then it will documents what the function returns, errors it may raise, and possibly other documentation as necessary. Often, the bottom of the document will contain examples.

Let's look at another example, the pandas drop function. This is used to drop rows or columns from a DataFrame. If you had a DataFrame call 'my_df', you would call this function by

```
my_df.drop(some arguments)
```

Unfortunately, we cannot simply call

```
help(drop)
```

because drop is not a function in base python. Instead, we must call

```
help(pd.DataFrame.drop)
```

because we need to specify that this is from pandas library (pd) and is applied to a DataFrame. If you're wondering why I'm capitalizing DataFrame as such, it is because that is a data type in the python pandas library. Without the capitalization, it had no meaning.

```
In [9]: help(pd.DataFrame.drop)

Help on function drop in module pandas.core.frame:

drop(self, labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='ra
    Drop specified labels from rows or columns.

    Remove rows or columns by specifying label names and corresponding
    axis, or by specifying directly index or column names. When using a
    multi-index, labels on different levels can be removed by specifying
    the level.

    Parameters
    ----------
    labels : single label or list-like
        Index or column labels to drop.
    axis : {0 or 'index', 1 or 'columns'}, default 0
        Whether to drop labels from the index (0 or 'index') or
        columns (1 or 'columns').
    index, columns : single label or list-like
        Alternative to specifying axis (``labels, axis=1``
        is equivalent to ``columns=labels``).

        .. versionadded:: 0.21.0
    level : int or level name, optional
        For MultiIndex, level from which the labels will be removed.
    inplace : bool, default False
        If True, do operation inplace and return None.
    errors : {'ignore', 'raise'}, default 'raise'
        If 'ignore', suppress error and only existing labels are
        dropped.

    Returns
    -------
    dropped : pandas.DataFrame

    Raises
    ------
    KeyError
        If none of the labels are found in the selected axis

    See Also
    --------
    DataFrame.loc : Label-location based indexer for selection by label.
    DataFrame.dropna : Return DataFrame with labels on given axis omitted
        where (all or any) data are missing.
    DataFrame.drop_duplicates : Return DataFrame with duplicate rows
        removed, optionally only considering certain columns.
```

```
Series.drop : Return Series with specified index labels removed.

Examples
--------
>>> df = pd.DataFrame(np.arange(12).reshape(3,4),
...                   columns=['A', 'B', 'C', 'D'])
>>> df
   A  B   C   D
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

Drop columns

>>> df.drop(['B', 'C'], axis=1)
   A   D
0  0   3
1  4   7
2  8  11

>>> df.drop(columns=['B', 'C'])
   A   D
0  0   3
1  4   7
2  8  11

Drop a row by index

>>> df.drop([0, 1])
   A  B   C   D
2  8  9  10  11

Drop columns and/or rows of MultiIndex DataFrame

>>> midx = pd.MultiIndex(levels=[['lama', 'cow', 'falcon'],
...                              ['speed', 'weight', 'length']],
...                      codes=[[0, 0, 0, 1, 1, 1, 2, 2, 2],
...                             [0, 1, 2, 0, 1, 2, 0, 1, 2]])
>>> df = pd.DataFrame(index=midx, columns=['big', 'small'],
...                   data=[[45, 30], [200, 100], [1.5, 1], [30, 20],
...                         [250, 150], [1.5, 0.8], [320, 250],
...                         [1, 0.8], [0.3,0.2]])
>>> df
                big     small
lama    speed   45.0    30.0
        weight  200.0   100.0
        length  1.5     1.0
cow     speed   30.0    20.0
```

```
        weight  250.0   150.0
        length  1.5     0.8
falcon  speed   320.0   250.0
        weight  1.0     0.8
        length  0.3     0.2

>>> df.drop(index='cow', columns='small')
                big
lama    speed   45.0
        weight  200.0
        length  1.5
falcon  speed   320.0
        weight  1.0
        length  0.3

>>> df.drop(index='length', level=1)
                big     small
lama    speed   45.0    30.0
        weight  200.0   100.0
cow     speed   30.0    20.0
        weight  250.0   150.0
falcon  speed   320.0   250.0
        weight  1.0     0.8
```

If you wanted to drop the column 'this one' from the DataFrame 'my_df', how would you do it?

```
In [16]: my_df = pd.DataFrame({"col1":[1,2,3], "col2": [4,5,6]})
         my_df

Out[16]:    col1  col2
         0     1     4
         1     2     5
         2     3     6

In [17]: my_df.drop(columns=['col1'], inplace=True)

In [18]: my_df

Out[18]:    col2
         0     4
         1     5
         2     6
```