Python, getting help and identifying variable types

HD Sheets, last modified 12/27/2023

In your career, you will wind up needing to learn and use many new packages and functions.

The first step is always to run the example files or a tutorial for a new package, and read all the documentation.

To then actually use the new tool on your data, you typically need to format your data to load into the tool, this is often the most tedious and annoying part of the process.

There are a huge variety of data storage formats in python, in addition to the four basic formats, people keep inventing new ones. Not all tools use the standard storage formats, you will run into new storage classes like "blobs" or even whole memory management systems like Apache Arrow, that are used to manage data from various modeling systems, not to mention distributed tools like Dask, Spark, Azures, AWS etc.

So when you are learning a new modeling or visualization tool, by working through tutorials or examples, you have to know how the input data is being formatted and stored.

Python has many ways of providing information about variables and functions.

What you need to learn from this notebook is how to make use of these, and then use them in future notebooks.

*Remember- Google is your friend! You will need to look things up constantly in this course, getting good at finding information using Google (or other tools) is a key part of this course. Some of the material below will be review for you, other concepts may be new, but you will need to use these ideas constantly during this course. *

Help on Functions

help(function_name), try help(print) and help(open)

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

```
# use with a member function

fo=open("foo.txt","w")
help(fo.flush)
```

```
Help on built-in function flush:

flush() method of _io.TextIOWrapper instance
    Flush write buffers, if applicable.

    This is not implemented for read-only and non-blocking streams.
```

```
help(open)
```

```
(the default is None has the same effect), or pass 'ignore' to ignore
errors. (Note that ignoring encoding errors can lead to data loss.)
See the documentation for codecs.register or run 'help(codecs.Codec)'
for a list of the permitted encoding error strings.

newline controls how universal newlines works (it only applies to text
mode). It can be None, '', '\n', '\r', and '\r\n'.  It works as
follows:

* On input, if newline is None, universal newlines mode is
  enabled. Lines in the input can end in '\n', '\r', or '\r\n', and
  these are translated into '\n' before being returned to the
  caller. If it is '', universal newline mode is enabled, but line
  endings are returned to the caller untranslated. If it has any of
  the other legal values, input lines are only terminated by the given
  string, and the line ending is returned to the caller untranslated.

* On output, if newline is None, any '\n' characters written are
  translated to the system default line separator, os.linesep. If
  newline is '' or '\n', no translation takes place. If newline is any
  of the other legal values, any '\n' characters written are translated
  to the given string.

If closefd is False, the underlying file descriptor will be kept open
when the file is closed. This does not work when a file name is given
and must be True in that case.

A custom opener can be used by passing a callable as *opener*. The
underlying file descriptor for the file object is then obtained by
calling *opener* with (*file*, *flags*). *opener* must return an open
file descriptor (passing os.open as *opener* results in functionality
similar to passing None).

open() returns a file object whose type depends on the mode, and
through which the standard file operations such as reading and writing
are performed. When open() is used to open a file in a text mode ('w',
'r', 'wt', 'rt', etc.), it returns a TextIOWrapper. When used to open
a file in a binary mode, the returned class varies: in read binary
mode, it returns a BufferedReader; in write binary and append binary
modes, it returns a BufferedWriter, and in read/write mode, it returns
a BufferedRandom.

It is also possible to use a string or bytearray as a file for both
reading and writing. For strings StringIO can be used like a file
opened in a text mode, and for bytes a BytesIO can be used like a file
opened in a binary mode.
```

Docstrings

Help() is returning the docstring for the function.

If you create functions for your work, always include a docstring

THis is a string inside triple quotes that the help function will return when you call help() on the function

```
def square_me(x):
    """  square_me(x)
    returns x squared
    """
    return x**2


square_me(5)
```

```
25
```

```
help(square_me)
```

```
Help on function square_me in module __main__:

square_me(x)
    square_me(x)
    returns x squared
```

Lists and getting information about variables

lists are one of the native data structures in Python and one of the most heavily used

We will create one and then try to get some information from python about it

When you see examples of code in books or online, they often don't tell you what the data structure or form is. To understand what code is doing and to then use the ideas in your own work, you need to understand how it works.

If you go to use the ideas in an example or piece of code, you need to understand the data structure you are dealing with, you will need to format input data properly to input it into a function, or to replicate an example calculation.

these are some methods to learn about variables

```
w=[1.4, 2.1, 3.3 ,1.2]


# note that the output appears in the Help window
w?
```

What did w? do for you? Notice that in the Jupyter notebook, the output of this request appears in a separate window you can close after reading.

Below is a list of lists, what does w2? show you?

```
w2=[[1,2,3],[4,3]]


type(w2)
```

```
     list
```

```
w2?

# notice that in Google Colab, the w2? output appears in a Help window of the browser, rather than in the notebook.
```

Question/Action

When I want you to modify code (an action) or answer a question, I will put in a cell like this one. Add a cell below it if there isn't one already there, and answer the question, or write the code in.

Question: In the cells above, we created a list w. What variable type is each element in w?

We also created a list w2. What is each element of w2?

Answer the question here

```
##Question/Action

# how would you reference or access the value 3.1 in w?
print(w[1])

# How would you reference the value 4 in w2
print(w2[1][0])

# Remember Python starts indexing at 0,  unlike R which starts at 1
```

```
     2.1
     4
```

```
# what type of thing is this?

a={1,2,3,4}
type(a)
```

```
     set
```

Question/Action

Build a set with the elements 1,2,3,1,4,2

What happens?

Sets are useful for testing membership, it is faster to search a set than a list

```
set1 = {1,2,3,1,4,2}
set1
```

```
{1, 2, 3, 4}
```

```
# how about this one
b=(5,6,7,8)
type(b)
```

```
tuple
```

```
b?
```

```
# one more basic data type

c={'name':'Sue','age':35,'Favorite color':'red, no blue!'}
```

```
c?
```

Question/Action

w, a,b,c are examples of the four basic advanced storage types in python

In the cell below, explain what the 4 types are. For each indicate whether they are mutable or not, and if they are iterable or not

What does it mean to be iterable? What does it mean to be mutable?

Remember, Google is your friend...

Also note that each of these storage types can hold entire items from the other classes, w2 was a list of lists

Insert Student Response here

- Set, List, Dictionary = Mutable
- Set, List, Tupble, Dictionary = Iterable

Double-click (or enter) to edit

Question/Action show an example of iteration or realization of the elements in a list here

```
for item in w:
  print(item)
```

```
1.4
2.1
3.3
1.2
```

Getting a list of the atributes of an object

In Python, everything is an object, or an instance of a class. Each has a number of different attributes.

You can see the attributes using the dir(variable_name) function

Many of these are functions (aka methods or member functions of the class), others are variables.

Any name with no double leading underline __, is a callable member function

I find dir very helpful in understanding what I am working with and what member functions might be available and helpful

```
dir(w)
```

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

```
print(w.__class__)
print(w.__subclasshook__)
```

```
<class 'list'>
<built-in method __subclasshook__ of type object at 0x569904065b00>
```

```
help(w.sort)
```

```
Help on built-in function sort:

sort(*, key=None, reverse=False) method of builtins.list instance
    Sort the list in ascending order and return None.

    The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
    order of two equal elements is maintained).

    If a key function is given, apply it once to each list item and sort them,
    ascending or descending, according to their function values.

    The reverse flag can be set to sort in descending order.
```

When w is an object, we can use the help function on it, to get a bunch of useful information about it, this may give us a lot of ideas about how to work with it.

When you are looking at example codes, use these functions to help you understand what is going on in the example

In working with an example, you are trying to understand how the example works, so you can modify the ideas and use them in your own work.

```
help(w)
```

```
Help on list object:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
```

There is a basic type() function like in R, which is some help

```
type(w)
```

```
list
```

```
# Python also has an id function,  this is a major help in some situations

# What does id actually do?
```

```
id(w)
```

```
140592014035904
```

```
d=w
print(d,w)
```

```
[1.4, 2.1, 3.3, 1.2] [1.4, 2.1, 3.3, 1.2]
```

```
e=w.copy()
print(e)
```

```
[1.4, 2.1, 3.3, 1.2]
```

```
id(d)
```

```
140592014035904
```

```
id(w)
```

```
140592014035904
```

```
id(e)
```

```
140590956113600
```

Look at the use of id() on w, d and e

Are w and d different ideas or not? Does e have a different id? What is happening here

Student input required here

Create a summary for later reference, what are the approaches shown in this example to getting information about a variable?

Why is it so important to understand the difference between d=w and e=w.copy?

```
e[2]=0
print(e)
print(w)
```

```
[1.4, 2.1, 0, 1.2]
[1.4, 2.1, 3.3, 1.2]
```

```
d[2]=-11
print(d)
print(w)
print(e)
```

```
[1.4, 2.1, -11, 1.2]
[1.4, 2.1, -11, 1.2]
[1.4, 2.1, 0, 1.2]
```

Student Input Required Here

Explain what is going on with w, d and e here. Explain how id() helps your understand this issue.

Student input required here

Create a summary for later reference, what are the approaches shown in this example to getting information about a variable?

- help() -> Doc string (Creators note about the function, in-builds or created)
- dir() -> attrubites(information about the object) or member functions/methods

Pandas and Numpy

These are two key packages used in Python to provide data storage classes that handle matrices (Numpy) and dataframes (Pandas)

Matrices act like the matrices in linear algebra and Numpy implements most standard linear algebra operations, it actually works much like Matlab or Octave. Data in a matrix is homogeneous in data type (often float or complex types)

Pandas implement dataframes that act like R dataframes, or somewhat like tabular data in a database

Numpy and Pandas have to be installed using conda or pip, and then they have to be imported into a Python program, as below

The online manuals for numpy and pandas are really good...

https://numpy.org/doc/stable/user/absolute_beginners.html

https://pandas.pydata.org/

Pandas makes use of Numpy storage classes, as does DASK

If you are taking DAT 512, we will take the time to really learn Numpy and Pandas

If you just in DAT 514 (but not concurrently in DAT 512), you will pick up Numpy and Pandas on the fly.

Most of the sci-kit learn tools we will using in DAT 514 use both Numpy and Pandas, and it is easy to move between Pandas storage classes and Numpy.

Below are some quick looks at Numpy and Pandas storage classes

```
import pandas as pd
import numpy as np

# note the use of import..as...
# this allows you to assign the aliases pd, and np for Pandas and Numpy
# this is a wide-spread practice, using these aliases consistently makes a bit easier


# there are other ways to do this, but here's a simple approach

x=np.array([[1,2,3],[4,5,6],[7,8,9]])
```

Use the tactics you have for learning info about python objects on this np array x

student input required below:

```
#There are many ways to create Pandas data frames,  I'm just going to convert my np matrix to a dataframe

xp=pd.DataFrame(x, columns=['A','B','C'])
```

Use the tactics you have for learning info about python objects on this np array x

student input required below:

Numpy and Pandas both have a bunch of data loading tools, we'll see them over the course of the semester

In DAT 514, we will seem np and pd used heavily, but the emphasis in DAT 514 is building predictive models

In DAT 512, we will spend some substantial time on learning np and pd pretty well

If you are in DAT 514 and really want to learn more on np and pd, see the courses on kaggle, and or do a search on pandas in the O'Reilly platform.