08.1_Scripts_Modules

October 10, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- Scripts
- Modules
- Packages

1.0.1 Two working modes: interactively and running scripts

- Interactive working mode (interpreter (code cell in a Jupyter Notebook))
  - You can use the interpreter to build up and test pieces of code until you get them working to your liking, at which point you can save them to the text file/script.
- Running scripts:
  - Scripts are useful when you want to permanently save some code with an eye for reuse later.

Complementary: using both an interpreter and scripts together is a common use pattern for scientific programming in Python.

2 What is a script?

- write the code in text files
  - using a text editor
  - Integrated Development Environment (IDE)
    - more features such as build automation, code highlighting, testing and debugging.

        * examples: PyCharm, Spyder, visual studio
- The file has the **.py** extension

2.0.1 Writing and Running Python Scripts

- Create a new file from the Jupyter directory called `hello.py` and enter the following into this file

```
print('Hello World!')
```

- Running the script `hello.py`
  - Shell (Windows: Git BASH, Mac: terminal):

```
python hello.py
```

  - Python interpreter:

```
>>> exec(open("hello.py").read())
```

[1]:
```
exec(open("hello11.py").read())
```

```
hello!
```

3 What is a Module?

- More organized "script"
  - A module is a set of **functions and data structures**.
- A module is a file containing Python code that can be **reused** in other Python code files.
- The file of a Python module has the **.py** extension.
- A module is often reused by `import` statements.

3.1 Standard Python Modules

Python comes with a rich set of standard modules (learn more):

- Numeric and Mathematical Modules
  - numbers — Numeric abstract base classes
  - math — Mathematical functions
  - cmath — Mathematical functions for complex numbers
  - decimal — Decimal fixed point and floating point arithmetic
  - fractions — Rational numbers
  - random — Generate pseudo-random numbers
  - statistics — Mathematical statistics functions
- Functional Programming Modules
  - itertools — Functions creating iterators for efficient looping
  - functools — Higher-order functions and operations on callable objects
  - operator — Standard operators as functions

3.2 Using Modules

- getting the whole module: `import module_name`

- only getting a specific function/class/attribute: `from module_name import something`

[2]:
```
import math
```

Use `math.` tab to inspect all the objects (functions, variables, etc) in the current namespace

[3]:
```
math.
```

```
  Cell In[3], line 1
    math.
         ^
SyntaxError: invalid syntax
```

[4]:
```
math.isinf?
```

```
Signature: math.isinf(x, /)
Docstring: Return True if x is a positive or negative infinity, and False␣
 ↪otherwise.
Type:      builtin_function_or_method
```

[5]:
```
help(math.isinf)
```

```
Help on built-in function isinf in module math:

isinf(x, /)
    Return True if x is a positive or negative infinity, and False otherwise.
```

[6]:
```
dir(math) # used to find out which names a module defines
```

[6]:
```
['__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'cbrt',
 'ceil',
 'comb',
```

```
'copysign',
'cos',
'cosh',
'degrees',
'dist',
'e',
'erf',
'erfc',
'exp',
'exp2',
'expm1',
'fabs',
'factorial',
'floor',
'fmod',
'frexp',
'fsum',
'gamma',
'gcd',
'hypot',
'inf',
'isclose',
'isfinite',
'isinf',
'isnan',
'isqrt',
'lcm',
'ldexp',
'lgamma',
'log',
'log10',
'log1p',
'log2',
'modf',
'nan',
'nextafter',
'perm',
'pi',
'pow',
'prod',
'radians',
'remainder',
'sin',
'sinh',
'sqrt',
'tan',
'tanh',
```

```
 'tau',
 'trunc',
 'ulp']
```

Use the form `module.function_name` to call a function that lives inside the module.

[7]:
```
math.sqrt(4)
```

[7]:
```
2.0
```

Use the form `module.attribute` to call an attribute that lives inside the module.

[8]:
```
math.pi
```

[8]:
```
3.141592653589793
```

Use an alias to rename the module upon import, which is handy when you have a long module name and want to save some typing.

[9]:
```
import math as m
```

[10]:
```
m.pi
```

[10]:
```
3.141592653589793
```

[11]:
```
pi = 10
```

[12]:
```
m.pi
```

[12]:
```
3.141592653589793
```

`from module_name import something`

[13]:
```
dir() # get a list of names comprising the attributes of the current namespac
```

[13]:
```
['In',
 'Out',
 '_',
 '_10',
 '_12',
 '_6',
 '_7',
 '_8',
 '__',
 '___',
 '__builtin__',
 '__builtins__',
 '__doc__',
 '__loader__',
```

```
 '__name__',
 '__package__',
 '__session__',
 '__spec__',
 '_dh',
 '_i',
 '_i1',
 '_i10',
 '_i11',
 '_i12',
 '_i13',
 '_i2',
 '_i3',
 '_i4',
 '_i5',
 '_i6',
 '_i7',
 '_i8',
 '_i9',
 '_ih',
 '_ii',
 '_iii',
 '_oh',
 'exit',
 'get_ipython',
 'm',
 'math',
 'open',
 'pi',
 'quit']
```

[14]:
```
from math import floor
```

[15]:
```
help(floor)
```

```
Help on built-in function floor in module math:

floor(x, /)
    Return the floor of x as an Integral.

    This is the largest integer <= x.
```

[16]:
```
dir()
```

[16]:
```
['In',
 'Out',
```

```
'_',
'_10',
'_12',
'_13',
'_6',
'_7',
'_8',
'__',
'___',
'__builtin__',
'__builtins__',
'__doc__',
'__loader__',
'__name__',
'__package__',
'__session__',
'__spec__',
'_dh',
'_i',
'_i1',
'_i10',
'_i11',
'_i12',
'_i13',
'_i14',
'_i15',
'_i16',
'_i2',
'_i3',
'_i4',
'_i5',
'_i6',
'_i7',
'_i8',
'_i9',
'_ih',
'_ii',
'_iii',
'_oh',
'exit',
'floor',
'get_ipython',
'm',
'math',
'open',
'pi',
'quit']
```

[17]:
```
floor(4.1)
```

[17]:
```
4
```

[18]:
```
math.floor(4.1)
```

[18]:
```
4
```

[19]:
```
from math import * # import everything in the math module, not recommended
```

[20]:
```
dir()
```

[20]:
```
['In',
 'Out',
 '_',
 '_10',
 '_12',
 '_13',
 '_16',
 '_17',
 '_18',
 '_6',
 '_7',
 '_8',
 '__',
 '___',
 '__builtin__',
 '__builtins__',
 '__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__session__',
 '__spec__',
 '_dh',
 '_i',
 '_i1',
 '_i10',
 '_i11',
 '_i12',
 '_i13',
 '_i14',
 '_i15',
 '_i16',
 '_i17',
 '_i18',
 '_i19',
```

```
'_i2',
'_i20',
'_i3',
'_i4',
'_i5',
'_i6',
'_i7',
'_i8',
'_i9',
'_ih',
'_ii',
'_iii',
'_oh',
'acos',
'acosh',
'asin',
'asinh',
'atan',
'atan2',
'atanh',
'cbrt',
'ceil',
'comb',
'copysign',
'cos',
'cosh',
'degrees',
'dist',
'e',
'erf',
'erfc',
'exit',
'exp',
'exp2',
'expm1',
'fabs',
'factorial',
'floor',
'fmod',
'frexp',
'fsum',
'gamma',
'gcd',
'get_ipython',
'hypot',
'inf',
'isclose',
```

```
 'isfinite',
 'isinf',
 'isnan',
 'isqrt',
 'lcm',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'm',
 'math',
 'modf',
 'nan',
 'nextafter',
 'open',
 'perm',
 'pi',
 'pow',
 'prod',
 'quit',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
 'tau',
 'trunc',
 'ulp']
```

[21]:
```
pi
```

[21]:
```
3.141592653589793
```

[22]:
```
import math
```

[23]:
```
math.pi
```

[23]:
```
3.141592653589793
```

3.2.1 Issues with `from math import *`

While this approach might seem like a good idea, it can create namespace clashes which arise if the module has objects with names that are identitical to those in the current namespace. In other

words if we already had a `floor` object in our namespace, it is now overwritten and points to the object in the math module and not the original `floor` object.

So using the explicit module name or alias to protect the namespace is generally **good practice**. It also lets us reuse the same name across different modules.

3.2.2 Further learning about python standard modules

- There are many more python standard modules that are very useful.
  - Follow the following tutorials/documentations to learn more about them:
    - Brief Tour of the Standard Library
    - Python 3 Module of the Week
    - Python - Built-in Modules from tutorialsteacher
- These standard python modules are building blocks of many external modules and projects

[24]:
```
import os #python standard module for operating system interface
```

[25]:
```
os.getcwd() # Current Directory
```

[25]:
```
'/Users/wk0110/My Drive (weikang9009@gmail.com)/teaching/Intro to
Python/2023_Fall/geog5560_2023Fall/notebooks'
```

[26]:
```
os.listdir()  #Return a list containing the names of the files in the directory.
```

[26]:
```
['hello.py',
 '02.2_Functions.ipynb',
 '05.1_Strings.ipynb',
 '.DS_Store',
 '15.1_Mapping.ipynb',
 '01.1_Introduction.ipynb',
 '12.1_geopandas.ipynb',
 '06.1_Lists.ipynb',
 '02.1_Program-Variables-Operators.ipynb',
 '08.1_Python_Ecosystem.ipynb',
 '11.1_Matplotlib.ipynb',
 '01.2_Installation-Notebook-GitHub.ipynb',
 'ex1.csv',
 '08.2_Iteration(2).ipynb',
 'hw4.py',
 '10.2_Numpy(1).ipynb',
 'pics',
 '07.1_Sets_Dictionaries.ipynb',
 'ex5.csv',
 '__pycache__',
 '11.2_pandas.ipynb',
 'hello11.py',
 '04.1_Conditionals_Strings.ipynb',
 '04_Git.ipynb',
```

```
 '10.1_OOP(3).ipynb',
 'untitled1.txt',
 '8.1_Final_Project_Template.ipynb',
 '09.2_OOP(2).ipynb',
 '03.1_ScalarDataTypes.ipynb',
 '.ipynb_checkpoints',
 '10.2_Numpy(2).ipynb',
 '09.1_OOP(1).ipynb',
 '06.2_Lists_Tuples-Copy1.ipynb',
 '04.2_Strings_Iteration.ipynb',
 'data',
 '05.2_Strings_Lists.ipynb',
 '06.2_Lists_Tuples.ipynb',
 '08.1_Final_Project.ipynb',
 '08.2_Functions(2).ipynb',
 '08.1_Scripts_Modules.ipynb']
```

[27]:
```
os.listdir('/Users/wk0110/My Drive/teaching')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[27], line 1
----> 1 os.listdir('/Users/wk0110/My Drive/teaching')

FileNotFoundError: [Errno 2] No such file or directory: '/Users/wk0110/My Drive
 ↪teaching'
```

3.3 Writing our own module

We will create a module and include the function that was used to `Check whether two words (strings) are the reverse of each other` (Part B of HW4) in the module. Our module will be named `hw4.py` and the function will be called `is_reverse` and take two arguments:

- string1
- string2

It will return `True` or `False`.

From the Jupyter Notebook Dashboard, click "New"-"Text File" to create the file `hw4.py` so that its contents are as follows (You may copy your solution to Part B of HW4 to this file instead)::

```
def is_reverse(string1, string2):
    """compare two words (strings) and return True if one of the words is the reverse of the o

    Parameters
    ----------
    string1 : str
              A string containing one word
```

```
    string2 : str
              A string containing one word

    Return
    ------
            : bool
              True if string1 is the reverse of string2; Otherwise, False

    """
    length1 = len(string1)
    length2 = len(string2)
    if length1 != length2:
        return False

    for i in range(length1):
        if string1[i] != string2[-1-i]:
            return False
    return True
```

Importing the module gives access to its objects, using the `module.object` syntax

Now we can use our module by importing the function from the module:

[28]:
```
from hw4 import is_reverse
```

[29]:
```
dir()
```

[29]:
```
['In',
 'Out',
 '_',
 '_10',
 '_12',
 '_13',
 '_16',
 '_17',
 '_18',
 '_20',
 '_21',
 '_23',
 '_25',
 '_26',
 '_6',
 '_7',
 '_8',
 '__',
 '___',
 '__builtin__',
 '__builtins__',
```

```
'__doc__',
'__loader__',
'__name__',
'__package__',
'__session__',
'__spec__',
'_dh',
'_i',
'_i1',
'_i10',
'_i11',
'_i12',
'_i13',
'_i14',
'_i15',
'_i16',
'_i17',
'_i18',
'_i19',
'_i2',
'_i20',
'_i21',
'_i22',
'_i23',
'_i24',
'_i25',
'_i26',
'_i27',
'_i28',
'_i29',
'_i3',
'_i4',
'_i5',
'_i6',
'_i7',
'_i8',
'_i9',
'_ih',
'_ii',
'_iii',
'_oh',
'acos',
'acosh',
'asin',
'asinh',
'atan',
'atan2',
```

```
'atanh',
'cbrt',
'ceil',
'comb',
'copysign',
'cos',
'cosh',
'degrees',
'dist',
'e',
'erf',
'erfc',
'exit',
'exp',
'exp2',
'expm1',
'fabs',
'factorial',
'floor',
'fmod',
'frexp',
'fsum',
'gamma',
'gcd',
'get_ipython',
'hypot',
'inf',
'is_reverse',
'isclose',
'isfinite',
'isinf',
'isnan',
'isqrt',
'lcm',
'ldexp',
'lgamma',
'log',
'log10',
'log1p',
'log2',
'm',
'math',
'modf',
'nan',
'nextafter',
'open',
'os',
```

```
 'perm',
 'pi',
 'pow',
 'prod',
 'quit',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
 'tau',
 'trunc',
 'ulp']
```

[30]:
```
is_reverse?
```

```
Signature: is_reverse(string1, string2)
Docstring:
compare two words (strings) and return True if one of the words is the reverse␣
 ↪of the other

Parameters
----------
string1 : str
          A string containing one word
string2 : str
          A string containing one word

Return
------
        : bool
          True if string1 is the reverse of string2; Otherwise, False
File:      ~/My Drive (weikang9009@gmail.com)/teaching/Intro to Python/2023_Fall/
 ↪geog5560_2023Fall/notebooks/hw4.py
Type:      function
```

[31]:
```
is_reverse("Happy", "Happy")
```

[31]:
```
False
```

[32]:
```
is_reverse("Happy", "yppaH")
```

[32]:
```
True
```

Or we import the module:

[33]:
```
import hw4
```

[34]:
```
hw4.is_reverse("ha","ds")
```

[34]:
```
False
```

[35]:
```
hw4?
```

```
Type:        module
String form: <module 'hw4' from '/Users/wk0110/My Drive (weikang9009@gmail.com)/
 ↪teaching/Intro to Python/2023_Fall/geog5560_2023Fall/notebooks/hw4.py'>
File:        ~/My Drive (weikang9009@gmail.com)/teaching/Intro to Python/
 ↪2023_Fall/geog5560_2023Fall/notebooks/hw4.py
Docstring:   <no docstring>
```

[36]:
```
dir(hw4)
```

[36]:
```
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'is_reverse']
```

[37]:
```
hw4.is_reverse("Happy", "Happy")
```

[37]:
```
False
```

[38]:
```
hw4.is_reverse("Happy", "yppaH")
```

[38]:
```
True
```

3.3.1 Documenting your module with Docstrings

- It is good practice to document your modules so that users know what and how to use the objects in the module.
- A Python docstring is a string used to document a Python module, class, function or method, so programmers can understand what it does without having to read the details of the implementation.
- It is a common practice to generate online (html) documentation automatically from docstrings.
- Everything between the pair of *triple quotes* is our function's **docstring**.
- There are some standard conventions and numpy doc style for writing docstrings that provide consistency.

[39]:
```
hw4.is_reverse?
```

```
Signature: hw4.is_reverse(string1, string2)
Docstring:
compare two words (strings) and return True if one of the words is the reverse␣
↪of the other

Parameters
----------
string1 : str
          A string containing one word
string2 : str
          A string containing one word

Return
------
        : bool
          True if string1 is the reverse of string2; Otherwise, False
File:      ~/My Drive (weikang9009@gmail.com)/teaching/Intro to Python/2023_Fall/
↪geog5560_2023Fall/notebooks/hw4.py
Type:      function
```

[40]:
```
help(hw4.is_reverse)
```

```
Help on function is_reverse in module hw4:

is_reverse(string1, string2)
    compare two words (strings) and return True if one of the words is the
reverse of the other

    Parameters
    ----------
    string1 : str
              A string containing one word
    string2 : str
              A string containing one word

    Return
    ------
            : bool
              True if string1 is the reverse of string2; Otherwise, False
```

3.3.2 Locating your module

When issuing an `import` statement Python searches a list of directories:

- The directory containing the input script (or the current directory).

- PYTHONPATH (a list of directory names, with the same syntax as the shell variable PATH).
- The installation-dependent default (by convention including a site-packages directory, handled by the site module).

Over time you will likely be accumutating more of your own modules, and rather than continuing to copy them to new working directories (and have multiple copies of the same file on your system) you can create your own central directory to contain your Python modules. How you do this depends on what operating system you are on. The Python Documentation wesbite provide

- instructions for windows
- instructions for Unix type systems (including Mac)

4 Python Packages

- Suitable for a large application that includes many *modules*
- Allow for a hierarchical structuring of the *module* namespace using dot notation `..` In the same way that modules help avoid collisions between global variable names, packages help avoid collisions between module names.
- special file called `__init__.py` (which may be empty) tells Python that the directory is a Python package, from which modules can be imported.

4.1 Importing a package

Given this structure, if the pkg directory resides in a location where it can be found (in one of the directories contained in `sys.path`), you can refer to the two modules with dot notation (`pkg.mod1`, `pkg.mod2`) and `import` them with the syntax you are already familiar with:

```
import pkg.mod1, pkg.mod2
```

If `mod1` has a function `bar()` and `mod2` has a function `foo()`, we can import and call these functions as shown below:

```
import pkg.mod1, pkg.mod2
pkg.mod1.bar()
pkg.mod2.foo()
```

Or

```
from pkg.mod1 import bar
bar()
from pkg.mod2 import foo
foo()
```

[41]:
```
import scipy
```

[42]:
```
scipy.__file__ #location of the package on your computer
```

[42]:
```
'/Users/wk0110/anaconda3/lib/python3.11/site-packages/scipy/__init__.py'
```

[43]:
```
scipy.__version__ #version of the package
```

[43]:
```
'1.11.2'
```

[44]:
```
dir(scipy)
```

[44]:
```
['LowLevelCallable',
 '__version__',
 'cluster',
 'constants',
 'datasets',
 'fft',
 'fftpack',
 'integrate',
 'interpolate',
 'io',
 'linalg',
 'misc',
 'ndimage',
 'odr',
 'optimize',
 'show_config',
 'signal',
 'sparse',
 'spatial',
 'special',
 'stats',
 'test']
```

[45]:
```
from scipy import stats
```

[46]:
```
dir(stats)
```

[46]:
```
['BootstrapMethod',
 'CensoredData',
 'ConstantInputWarning',
 'Covariance',
 'DegenerateDataWarning',
 'FitError',
 'MonteCarloMethod',
 'NearConstantInputWarning',
 'PermutationMethod',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
```

```
'__package__',
'__path__',
'__spec__',
'_axis_nan_policy',
'_biasedurn',
'_binned_statistic',
'_binomtest',
'_boost',
'_censored_data',
'_common',
'_constants',
'_continuous_distns',
'_covariance',
'_crosstab',
'_discrete_distns',
'_distn_infrastructure',
'_distr_params',
'_entropy',
'_fit',
'_hypotests',
'_kde',
'_ksstats',
'_levy_stable',
'_mannwhitneyu',
'_morestats',
'_mstats_basic',
'_mstats_extras',
'_multicomp',
'_multivariate',
'_mvn',
'_odds_ratio',
'_page_trend_test',
'_qmc',
'_qmc_cy',
'_qmvnt',
'_rcont',
'_relative_risk',
'_resampling',
'_rvs_sampling',
'_sensitivity_analysis',
'_sobol',
'_statlib',
'_stats',
'_stats_mstats_common',
'_stats_py',
'_stats_pythran',
'_survival',
```

```
'_tukeylambda_stats',
'_variation',
'_warnings_errors',
'alexandergovern',
'alpha',
'anderson',
'anderson_ksamp',
'anglit',
'ansari',
'arcsine',
'argus',
'barnard_exact',
'bartlett',
'bayes_mvs',
'bernoulli',
'beta',
'betabinom',
'betaprime',
'biasedurn',
'binned_statistic',
'binned_statistic_2d',
'binned_statistic_dd',
'binom',
'binom_test',
'binomtest',
'boltzmann',
'bootstrap',
'boschloo_exact',
'boxcox',
'boxcox_llf',
'boxcox_normmax',
'boxcox_normplot',
'bradford',
'brunnermunzel',
'burr',
'burr12',
'cauchy',
'chi',
'chi2',
'chi2_contingency',
'chisquare',
'circmean',
'circstd',
'circvar',
'combine_pvalues',
'contingency',
'cosine',
```

```
'cramervonmises',
'cramervonmises_2samp',
'crystalball',
'cumfreq',
'describe',
'dgamma',
'differential_entropy',
'directional_stats',
'dirichlet',
'dirichlet_multinomial',
'distributions',
'dlaplace',
'dunnett',
'dweibull',
'ecdf',
'energy_distance',
'entropy',
'epps_singleton_2samp',
'erlang',
'expectile',
'expon',
'exponnorm',
'exponpow',
'exponweib',
'f',
'f_oneway',
'false_discovery_control',
'fatiguelife',
'find_repeats',
'fisher_exact',
'fisk',
'fit',
'fligner',
'foldcauchy',
'foldnorm',
'friedmanchisquare',
'gamma',
'gausshyper',
'gaussian_kde',
'genexpon',
'genextreme',
'gengamma',
'genhalflogistic',
'genhyperbolic',
'geninvgauss',
'genlogistic',
'gennorm',
```

```
'genpareto',
'geom',
'gibrat',
'gmean',
'gompertz',
'goodness_of_fit',
'gstd',
'gumbel_l',
'gumbel_r',
'gzscore',
'halfcauchy',
'halfgennorm',
'halflogistic',
'halfnorm',
'hmean',
'hypergeom',
'hypsecant',
'invgamma',
'invgauss',
'invweibull',
'invwishart',
'iqr',
'jarque_bera',
'johnsonsb',
'johnsonsu',
'kappa3',
'kappa4',
'kde',
'kendalltau',
'kruskal',
'ks_1samp',
'ks_2samp',
'ksone',
'kstat',
'kstatvar',
'kstest',
'kstwo',
'kstwobign',
'kurtosis',
'kurtosistest',
'laplace',
'laplace_asymmetric',
'levene',
'levy',
'levy_l',
'levy_stable',
'linregress',
```

```
'loggamma',
'logistic',
'loglaplace',
'lognorm',
'logrank',
'logser',
'loguniform',
'lomax',
'mannwhitneyu',
'matrix_normal',
'maxwell',
'median_abs_deviation',
'median_test',
'mielke',
'mode',
'moment',
'monte_carlo_test',
'mood',
'morestats',
'moyal',
'mstats',
'mstats_basic',
'mstats_extras',
'multinomial',
'multiscale_graphcorr',
'multivariate_hypergeom',
'multivariate_normal',
'multivariate_t',
'mvn',
'mvsdist',
'nakagami',
'nbinom',
'ncf',
'nchypergeom_fisher',
'nchypergeom_wallenius',
'nct',
'ncx2',
'nhypergeom',
'norm',
'normaltest',
'norminvgauss',
'obrientransform',
'ortho_group',
'page_trend_test',
'pareto',
'pearson3',
'pearsonr',
```

```
'percentileofscore',
'permutation_test',
'planck',
'pmean',
'pointbiserialr',
'poisson',
'poisson_means_test',
'power_divergence',
'powerlaw',
'powerlognorm',
'powernorm',
'ppcc_max',
'ppcc_plot',
'probplot',
'qmc',
'randint',
'random_correlation',
'random_table',
'rankdata',
'ranksums',
'rayleigh',
'rdist',
'recipinvgauss',
'reciprocal',
'rel_breitwigner',
'relfreq',
'rice',
'rv_continuous',
'rv_discrete',
'rv_histogram',
'rvs_ratio_uniforms',
'scoreatpercentile',
'sem',
'semicircular',
'shapiro',
'siegelslopes',
'sigmaclip',
'skellam',
'skew',
'skewcauchy',
'skewnorm',
'skewtest',
'sobol_indices',
'somersd',
'spearmanr',
'special_ortho_group',
'statlib',
```

```
'stats',
'studentized_range',
't',
'test',
'theilslopes',
'tiecorrect',
'tmax',
'tmean',
'tmin',
'trapezoid',
'trapz',
'triang',
'trim1',
'trim_mean',
'trimboth',
'truncexpon',
'truncnorm',
'truncpareto',
'truncweibull_min',
'tsem',
'tstd',
'ttest_1samp',
'ttest_ind',
'ttest_ind_from_stats',
'ttest_rel',
'tukey_hsd',
'tukeylambda',
'tvar',
'uniform',
'uniform_direction',
'unitary_group',
'variation',
'vonmises',
'vonmises_fisher',
'vonmises_line',
'wald',
'wasserstein_distance',
'weibull_max',
'weibull_min',
'weightedtau',
'wilcoxon',
'wishart',
'wrapcauchy',
'yeojohnson',
'yeojohnson_llf',
'yeojohnson_normmax',
'yeojohnson_normplot',
```

```
 'yulesimon',
 'zipf',
 'zipfian',
 'zmap',
 'zscore']
```

[47]:
```
stats.boxcox?
```

```
Signature: stats.boxcox(x, lmbda=None, alpha=None, optimizer=None)
Docstring:
Return a dataset transformed by a Box-Cox power transformation.

Parameters
----------
x : ndarray
    Input array to be transformed.

    If `lmbda` is not None, this is an alias of
    `scipy.special.boxcox`.
    Returns nan if ``x < 0``; returns -inf if ``x == 0 and lmbda < 0``.

    If `lmbda` is None, array must be positive, 1-dimensional, and
    non-constant.

lmbda : scalar, optional
    If `lmbda` is None (default), find the value of `lmbda` that maximizes
    the log-likelihood function and return it as the second output
    argument.

    If `lmbda` is not None, do the transformation for that value.

alpha : float, optional
    If `lmbda` is None and `alpha` is not None (default), return the
    ``100 * (1-alpha)%`` confidence  interval for `lmbda` as the third
    output argument. Must be between 0.0 and 1.0.

    If `lmbda` is not None, `alpha` is ignored.
optimizer : callable, optional
    If `lmbda` is None, `optimizer` is the scalar optimizer used to find
    the value of `lmbda` that minimizes the negative log-likelihood
    function. `optimizer` is a callable that accepts one argument:

    fun : callable
        The objective function, which evaluates the negative
        log-likelihood function at a provided value of `lmbda`

    and returns an object, such as an instance of
```

```
    `scipy.optimize.OptimizeResult`, which holds the optimal value of
    `lmbda` in an attribute `x`.

    See the example in `boxcox_normmax` or the documentation of
    `scipy.optimize.minimize_scalar` for more information.

    If `lmbda` is not None, `optimizer` is ignored.

Returns
-------
boxcox : ndarray
    Box-Cox power transformed array.
maxlog : float, optional
    If the `lmbda` parameter is None, the second returned argument is
    the `lmbda` that maximizes the log-likelihood function.
(min_ci, max_ci) : tuple of float, optional
    If `lmbda` parameter is None and `alpha` is not None, this returned
    tuple of floats represents the minimum and maximum confidence limits
    given `alpha`.

See Also
--------
probplot, boxcox_normplot, boxcox_normmax, boxcox_llf

Notes
-----
The Box-Cox transform is given by::

    y = (x**lmbda - 1) / lmbda,  for lmbda != 0
        log(x),                  for lmbda = 0

`boxcox` requires the input data to be positive.  Sometimes a Box-Cox
transformation provides a shift parameter to achieve this; `boxcox` does
not.  Such a shift parameter is equivalent to adding a positive constant to
`x` before calling `boxcox`.

The confidence limits returned when `alpha` is provided give the interval
where:

.. math::

    llf(\hat{\lambda}) - llf(\lambda) < \frac{1}{2}\chi^2(1 - \alpha, 1),

with ``llf`` the log-likelihood function and :math:`\chi^2` the chi-squared
function.

References
----------
```

```
G.E.P. Box and D.R. Cox, "An Analysis of Transformations", Journal of the
Royal Statistical Society B, 26, 211-252 (1964).

Examples
--------
>>> from scipy import stats
>>> import matplotlib.pyplot as plt

We generate some random variates from a non-normal distribution and make a
probability plot for it, to show it is non-normal in the tails:

>>> fig = plt.figure()
>>> ax1 = fig.add_subplot(211)
>>> x = stats.loggamma.rvs(5, size=500) + 5
>>> prob = stats.probplot(x, dist=stats.norm, plot=ax1)
>>> ax1.set_xlabel('')
>>> ax1.set_title('Probplot against normal distribution')

We now use `boxcox` to transform the data so it's closest to normal:

>>> ax2 = fig.add_subplot(212)
>>> xt, _ = stats.boxcox(x)
>>> prob = stats.probplot(xt, dist=stats.norm, plot=ax2)
>>> ax2.set_title('Probplot after Box-Cox transformation')

>>> plt.show()
File:      ~/anaconda3/lib/python3.11/site-packages/scipy/stats/_morestats.py
Type:      function
```

4.1.1 Installing python packages

- `pip`: `pip install scipy`
  - The Python Package Index (PyPI) is a repository of software for the Python programming language.
- `conda`: `conda install scipy`
  - Anaconda Packages

4.2 Further readings on creating a python package

Read the Real Python tutorial on Python packages to learn more about creating modules and packages.

Read the open source book Python Packages to learn more about the modern and efficient workflows for creating Python packages.

[ ]: