modules

January 14, 2021

1 Module 0 - Python Modules

A PDF version of this notebook is available at Module 0 - Python Modules

1.1 Modules

A module is a data type that contains multiple functions or methods meant to solve specific problems. Generally, we need to install the module prior to using it. Python uses its own module installation manager: `pip`. The statement to install a module that has not yet been installed is: `pip install modulename` and it needs to be done in a terminal prompt.

Anaconda and Google Colab contain most of the modules we need to use in this class and provides ease of use for many useful machine learning implementations. Once a module has been installed, we only need to import it to the environment. We do this **every time** we open an instance of Python. Example of importing a module is given below:

[1]:
```
# if we need a function under the math module we need to run the line below
import math
```

The word `math` is simply a label (think variable name) in Python's global namespace that points to some object in memory that is the `math` module.

[2]:
```
type(math)
```

[2]:
```
module
```

The `math` module contains many objects inside. These objects include functions, constants that we may need to use.

[3]:
```
## we can check the objects by using the dir function
dir(math)
```

[3]:
```
['__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
```

```
'asin',
'asinh',
'atan',
'atan2',
'atanh',
'ceil',
'copysign',
'cos',
'cosh',
'degrees',
'e',
'erf',
'erfc',
'exp',
'expm1',
'fabs',
'factorial',
'floor',
'fmod',
'frexp',
'fsum',
'gamma',
'gcd',
'hypot',
'inf',
'isclose',
'isfinite',
'isinf',
'isnan',
'ldexp',
'lgamma',
'log',
'log10',
'log1p',
'log2',
'modf',
'nan',
'pi',
'pow',
'radians',
'sin',
'sinh',
'sqrt',
'tan',
'tanh',
'tau',
'trunc']
```

The information regarding the objects inside a function can be listed using `modulename.__dict__`. The example below `math.__dict__` contains more information about each object, including if the object is a function, e.g., `'cos': <function math.cos>` or a constant, e.g., `'pi': 3.141592653589793`

[4]:
```
math.__dict__
```

[4]:
```
{'__doc__': 'This module is always available.  It provides access to
the\nmathematical functions defined by the C standard.',
 '__loader__': _frozen_importlib.BuiltinImporter,
 '__name__': 'math',
 '__package__': '',
 '__spec__': ModuleSpec(name='math', loader=<class
'_frozen_importlib.BuiltinImporter'>, origin='built-in'),
 'acos': <function math.acos>,
 'acosh': <function math.acosh>,
 'asin': <function math.asin>,
 'asinh': <function math.asinh>,
 'atan': <function math.atan>,
 'atan2': <function math.atan2>,
 'atanh': <function math.atanh>,
 'ceil': <function math.ceil>,
 'copysign': <function math.copysign>,
 'cos': <function math.cos>,
 'cosh': <function math.cosh>,
 'degrees': <function math.degrees>,
 'e': 2.718281828459045,
 'erf': <function math.erf>,
 'erfc': <function math.erfc>,
 'exp': <function math.exp>,
 'expm1': <function math.expm1>,
 'fabs': <function math.fabs>,
 'factorial': <function math.factorial>,
 'floor': <function math.floor>,
 'fmod': <function math.fmod>,
 'frexp': <function math.frexp>,
 'fsum': <function math.fsum>,
 'gamma': <function math.gamma>,
 'gcd': <function math.gcd>,
 'hypot': <function math.hypot>,
 'inf': inf,
 'isclose': <function math.isclose>,
 'isfinite': <function math.isfinite>,
 'isinf': <function math.isinf>,
 'isnan': <function math.isnan>,
 'ldexp': <function math.ldexp>,
 'lgamma': <function math.lgamma>,
 'log': <function math.log>,
```

```
 'log10': <function math.log10>,
 'log1p': <function math.log1p>,
 'log2': <function math.log2>,
 'modf': <function math.modf>,
 'nan': nan,
 'pi': 3.141592653589793,
 'pow': <function math.pow>,
 'radians': <function math.radians>,
 'sin': <function math.sin>,
 'sinh': <function math.sinh>,
 'sqrt': <function math.sqrt>,
 'tan': <function math.tan>,
 'tanh': <function math.tanh>,
 'tau': 6.283185307179586,
 'trunc': <function math.trunc>}
```

We can find help on a module (or any object) by using `help(modulename)`.

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

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(…)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(…)
        acosh(x)

        Return the inverse hyperbolic cosine of x.

    asin(…)
        asin(x)

        Return the arc sine (measured in radians) of x.

    asinh(…)
        asinh(x)
```

```
    Return the inverse hyperbolic sine of x.

atan(…)
    atan(x)

    Return the arc tangent (measured in radians) of x.

atan2(…)
    atan2(y, x)

    Return the arc tangent (measured in radians) of y/x.
    Unlike atan(y/x), the signs of both x and y are considered.

atanh(…)
    atanh(x)

    Return the inverse hyperbolic tangent of x.

ceil(…)
    ceil(x)

    Return the ceiling of x as an Integral.
    This is the smallest integer >= x.

copysign(…)
    copysign(x, y)

    Return a float with the magnitude (absolute value) of x but the sign
    of y. On platforms that support signed zeros, copysign(1.0, -0.0)
    returns -1.0.

cos(…)
    cos(x)

    Return the cosine of x (measured in radians).

cosh(…)
    cosh(x)

    Return the hyperbolic cosine of x.

degrees(…)
    degrees(x)

    Convert angle x from radians to degrees.

erf(…)
    erf(x)
```

```
        Error function at x.

    erfc(…)
        erfc(x)

        Complementary error function at x.

    exp(…)
        exp(x)

        Return e raised to the power of x.

    expm1(…)
        expm1(x)

        Return exp(x)-1.
        This function avoids the loss of precision involved in the direct
evaluation of exp(x)-1 for small x.

    fabs(…)
        fabs(x)

        Return the absolute value of the float x.

    factorial(…)
        factorial(x) -> Integral

        Find x!. Raise a ValueError if x is negative or non-integral.

    floor(…)
        floor(x)

        Return the floor of x as an Integral.
        This is the largest integer <= x.

    fmod(…)
        fmod(x, y)

        Return fmod(x, y), according to platform C.  x % y may differ.

    frexp(…)
        frexp(x)

        Return the mantissa and exponent of x, as pair (m, e).
        m is a float and e is an int, such that x = m * 2.**e.
        If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.
```

```
    fsum(…)
        fsum(iterable)

        Return an accurate floating point sum of values in the iterable.
        Assumes IEEE-754 floating point arithmetic.

    gamma(…)
        gamma(x)

        Gamma function at x.

    gcd(…)
        gcd(x, y) -> int
        greatest common divisor of x and y

    hypot(…)
        hypot(x, y)

        Return the Euclidean distance, sqrt(x*x + y*y).

    isclose(…)
        isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0) -> bool

        Determine whether two floating point numbers are close in value.

           rel_tol
               maximum difference for being considered "close", relative to the
               magnitude of the input values
            abs_tol
               maximum difference for being considered "close", regardless of
the
               magnitude of the input values

        Return True if a is close in value to b, and False otherwise.

        For the values to be considered close, the difference between them
        must be smaller than at least one of the tolerances.

        -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
        is, NaN is not close to anything, even itself.  inf and -inf are
        only close to themselves.

    isfinite(…)
        isfinite(x) -> bool

        Return True if x is neither an infinity nor a NaN, and False otherwise.

    isinf(…)
```

```
        isinf(x) -> bool

        Return True if x is a positive or negative infinity, and False
otherwise.

    isnan(…)
        isnan(x) -> bool

        Return True if x is a NaN (not a number), and False otherwise.

    ldexp(…)
        ldexp(x, i)

        Return x * (2**i).

    lgamma(…)
        lgamma(x)

        Natural logarithm of absolute value of Gamma function at x.

    log(…)
        log(x[, base])

        Return the logarithm of x to the given base.
        If the base not specified, returns the natural logarithm (base e) of x.

    log10(…)
        log10(x)

        Return the base 10 logarithm of x.

    log1p(…)
        log1p(x)

        Return the natural logarithm of 1+x (base e).
        The result is computed in a way which is accurate for x near zero.

    log2(…)
        log2(x)

        Return the base 2 logarithm of x.

    modf(…)
        modf(x)

        Return the fractional and integer parts of x.  Both results carry the
sign
        of x and are floats.
```

```
    pow(…)
        pow(x, y)

        Return x**y (x to the power of y).

    radians(…)
        radians(x)

        Convert angle x from degrees to radians.

    sin(…)
        sin(x)

        Return the sine of x (measured in radians).

    sinh(…)
        sinh(x)

        Return the hyperbolic sine of x.

    sqrt(…)
        sqrt(x)

        Return the square root of x.

    tan(…)
        tan(x)

        Return the tangent of x (measured in radians).

    tanh(…)
        tanh(x)

        Return the hyperbolic tangent of x.

    trunc(…)
        trunc(x:Real) -> Integral

        Truncates x to the nearest Integral toward 0. Uses the __trunc__ magic
method.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586
```

```
FILE
    (built-in)
```

1.1.1 *More on Importing modules*

We can run a statement such as

`import modulename`

We can also create an alias (a name easier to remember or write) for a module:

e.g. the Pandas module is generally imported with the `pd` alias

[6]:
```
import pandas as pd
```

The NumPy module is generally imported with the `np` alias:

[7]:
```
import numpy as np
```

If we need to access specific functions or objects from a module we need to use the following pattern:

`modulename.function()`

[8]:
```
math.sqrt(9)
```

[8]:
```
3.0
```

If we import the whole `math` module, simply writing `sqrt()` creates an exception (error). We need to specify `math.sqrt(9)`.

[9]:
```
sqrt(9)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-840f67a85afc> in <module>()
----> 1 sqrt(9)

NameError: name 'sqrt' is not defined
```

In the case that we use a function multiple times and do not want to write the module name every time we use the function, the following pattern is preferred:

`from modulename import function`

[10]:
```
from math import sqrt
```

[11]:
```
sqrt(9)
```

[11]: 3.0

We can import all objects from a module

[12]:
```
from math import *
```

In this case we can use all of the functions within the `math` module without the need to specify `math.sqrt()`. E.g.,

[13]:
```
sin(0)
```

[13]: 0.0

The following import variants are valid:

- `import math`
- `from math import sqrt, abs`
- `from math import *`
- `import math as r_math`
- `from math import sqrt as r_sqrt`

There is not much of an advantage of using a variant as opposed to other in terms of computational speed.