# 4_modules

October 17, 2023

## 1 Modules and namespaces

### 1.1 Modules

Most of the functionality in Python is provided by *modules*. The Python Standard Library is a large collection of modules that provides *cross-platform* implementations of common facilities such as access to the operating system, file I/O, string management, network communication, math, web-scraping, text manipulation, machine learning and much more.

To use a module in a Python program it first has to be imported. A module can be imported using the `import` statement. For example, to import the module `math`, which contains many standard mathematical functions, we can do:

[2]:
```
import math
import numpy

x = math.sqrt(2 * math.pi)
x = numpy.sqrt(2 * numpy.pi)


print(x)
```

```
2.5066282746310002
```

This includes the whole module and makes it available for use later in the program. Alternatively, we can chose to import all symbols (functions and variables) in a module so that we don't need to use the prefix "`math.`" every time we use something from the `math` module:

[4]:
```
from math import *

x = cos(2 * pi)

print(x)
```

```
1.0
```

This pattern can be very convenient, but in large programs that include many modules it is often a good idea to keep the symbols from each module in their own namespaces, by using the `import math` pattern. This would eliminate potentially confusing problems.

## 1.2 Namespaces

A namespace is an identifier used to organize objects, e.g. the methods and variables of a module. The prefix `math.` we have used in the previous section is such a namespace. You may also create your own namespace for a module.

[5]:
```
import math as m

x = m.sqrt(2)

print(x)
```

```
1.4142135623730951
```

You may also only import specific functions of a module.

[6]:
```
from math import sinh as mysinh
```

## 1.3 Contents of a module

Once a module is imported, we can list the symbols it provides using the `dir` function:

[11]:
```
import math

print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb',
'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd',
'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow',
'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau',
'trunc']
```

And using the function `help` we can get a description of each function (almost .. not all functions have docstrings, as they are technically called, but the vast majority of functions are documented this way).

[12]:
```
help(math.log)
```

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

log(…)
    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.
```

[13]:
```
math.log(10)
```

[13]:
```
2.302585092994046
```

[13]:
```
math.log(8, 2)
```

[13]:
```
3.0
```

We can also use the `help` function directly on modules: Try

`help(math)`

Some very useful modules form the Python standard library are `os`, `sys`, `math`, `shutil`, `re`, `subprocess`, `multiprocessing`, `threading`.

A complete lists of standard modules for Python 3 is available at http://docs.python.org/3/library/ .