# ASTR 310
# Computing in Astronomy

Spring 2024

Lecture 11: Programming VIII: Python: modules

```
            f.close()
        except IOError:
            print "Could not read file."
            sys.exit(1)
        print "done."

    def WriteToFile (self, file):
        print "Writing "+file+" ..."
        try:
            f = open(file, 'w')
            for name in self.state.keys():
                if not self.state[name][4]:
                    if self.state[name][1] != "":
                        f.write("%s    %s\n" % (name, self.state[name][1]))
                    else:
                        f.write("%s    %s\n" % (name, '""'))
            f.close()
        except IOError:
            print "Could not write file."
            sys.exit(1)
        print "done."

#---------------------------------------------------------------------------
```

Dr Lisa Young
University of Illinois

# Namespaces

Variable, function, and class names have a **namespace** within which they are defined.

Importing a Python module brings its own namespace into a program. Options:

- Incorporate into current namespace

```
from math import *
from math import cos, sin, tan
```

- Incorporate but rename

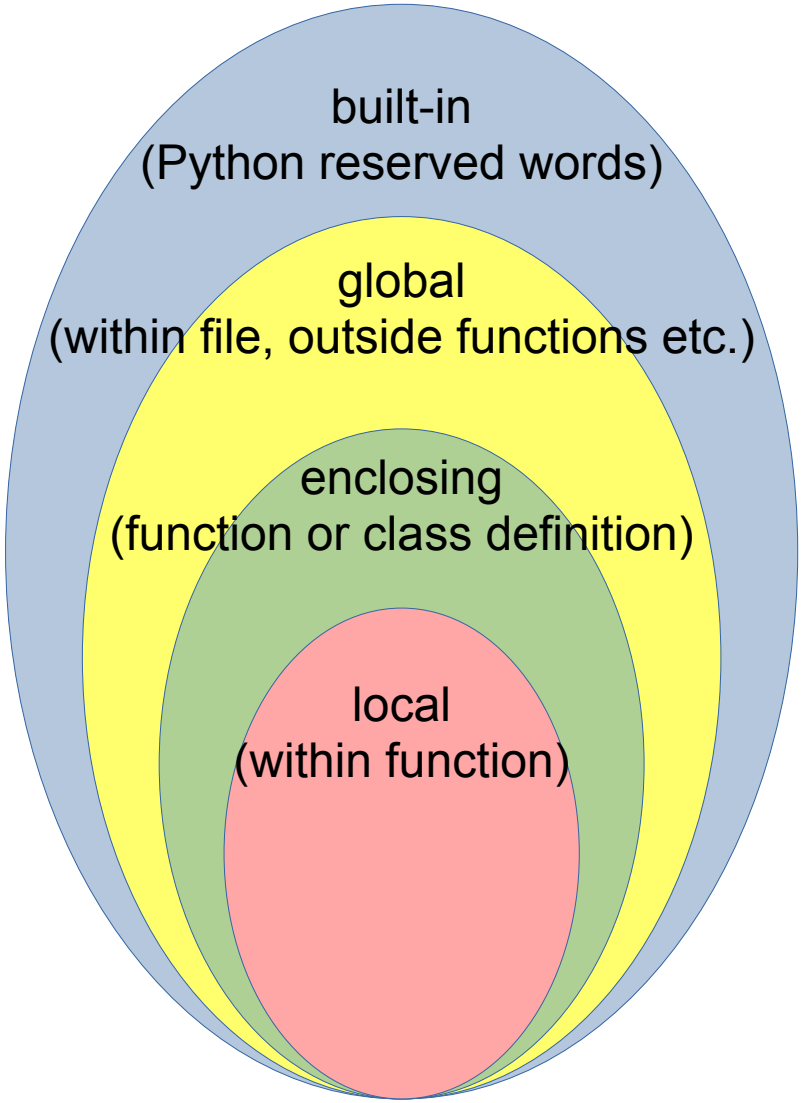
```
from math import cos as mycos
```

- Define a separate namespace

```
import math
```

- Rename separate namespace

```
import math as m
```



# Python standard library

| | |
|---|---|
| `copy` | shallow and deep copy operations (e.g. on lists) |
| `math, cmath` | basic (real/complex) math operations and constants |
| `string` | string operations, esp. string formatting |
| `struct` | convert byte data to/from binary form |
| `os` | operating system interfaces (environment variables, user ID, current directory, etc.) |
| `os.path` | path name manipulation (e.g. extract file name from a path) |
| `glob` | Unix path name pattern expansion |
| `shutil` | high-level file operations (move, copy, remove files) |
| `subprocess` | subprocess management (launch programs and get data from them) |
| `sys` | system-specific parameters and functions (e.g. command-line arguments) |

… and many more (e.g. modules to do pattern matching, Internet access, …)

See https://docs.python.org/3/library/index.html for a full list.

# Exercise 1: random stuff

The `random` module in the standard library provides random number-related functions, particularly

| | |
|---|---|
| `random.randint(a, b)` | return a random integer between `a` and `b`, inclusive |
| `random.random()` | return a random float in the range [0,1) |
| `random.shuffle(S)` | randomly shuffle the sequence `S` in place |
| `random.choice(S)` | return a random choice from the sequence `S` |

Write Python code to:

- create a sorted random list of 100 floats between 0 and $2\pi$
- generate a six-digit random integer code (convert to string and pad with zeros if it is less than 100000)
- generate a ten-character random string containing characters from a set including upper- and lowercase letters, digits, and '#!$@*%'. You may like to check helpful information in the documentation for the `string` library.

# Exercise 2: time

The `time` module in the standard library provides, unsurprisingly, time-related functions, in particular:

| | |
|---|---|
| `time.time()` | return the number of seconds since January 1, 1970 00:00:00 UTC (known as the **epoch**) |
| `time.localtime()` | return a `time_struct` object containing the local time |
| `time.asctime(ts)` | given a `time_struct` object `ts`, return a string-formatted date and time |
| `time.perf_counter()` | return seconds of a highly accurate counter |
| `time.sleep(n)` | pause for `n` seconds |

Download the Sieve of Eratosthenes module (`sieve.py`) from the course Canvas page. This contains a single function, `getprimes(n)`, which returns a list of all primes up to `n`. You can import this function into your notebook with `import`.

Write a Python function to accept a value of n, then time a call to `getprimes(n)` and return a tuple containing the output of the function and the elapsed time.

If you get bored: make your timing function work with generic functions with arbitrary arguments.

# Exercise 3: directory listing

Write a standalone Python program (**not** a Jupyter notebook) that uses the `sys` and `os` modules to generate a directory tree listing. The program should take one argument, the name of a directory, and recursively print the names of all files and directories below it. Try to produce output like the following:

```
$ python dirtree.py foo
foo
|-- a.txt
|-- b.txt
|-- code
|    |-- a.py
|    |-- b.py
|    |-- docs
|    |    |-- a.txt
|    |    |-- b.txt
|    |-- x.py
|-- z.txt
```

When you're done, please copy-paste your programs and their output into your notebook and upload the pdf of the whole thing to Canvas for today's exercise.

**Hints:** `sys.argv`, `os.listdir`, and `os.path.isdir` might be helpful. Also, you may find it easiest to write a recursive function. Try just traversing the directory structure before you print the fancy stuff (`|--` and spacing).