10. Brief Tour of the Standard Library
****

10.1. Operating System Interface
================================

The "os" module provides dozens of functions for interacting with the
operating system:

```
>>> import os
>>> os.getcwd() # Return the current working directory
'C:\\Python314'
>>> os.chdir('/server/accesslogs') # Change current working directory
>>> os.system('mkdir today') # Run the command mkdir in the system shell
0
```

Be sure to use the "import os" style instead of "from os import *".
This will keep "os.open()" from shadowing the built-in "open()"
function which operates much differently.

The built-in "dir()" and "help()" functions are useful as interactive
aids for working with large modules like "os":

```
>>> import os
>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the module's docstrings>
```

For daily file and directory management tasks, the "shutil" module
provides a higher level interface that is easier to use:

```
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
'archive.db'
>>> shutil.move('/build/executables', 'installdir')
'installdir'
```

10.2. File Wildcards
===================

The "glob" module provides a function for making file lists from
directory wildcard searches:

```
>>> import glob
>>> glob.glob('*.py')
['primes.py', 'random.py', 'quote.py']
```

10.3. Command Line Arguments
============================

Common utility scripts often need to process command line arguments.
These arguments are stored in the "sys" module's *argv* attribute as a

list. For instance, let's take the following "demo.py" file:

```
# File demo.py
import sys
print(sys.argv)
```

Here is the output from running "python demo.py one two three" at the command line:

```
['demo.py', 'one', 'two', 'three']
```

The "argparse" module provides a more sophisticated mechanism to process command line arguments. The following script extracts one or more filenames and an optional number of lines to be displayed:

```
import argparse

parser = argparse.ArgumentParser(
prog='top',
description='Show top lines from each file')
parser.add_argument('filenames', nargs='+')
parser.add_argument('-l', '--lines', type=int, default=10)
args = parser.parse_args()
print(args)
```

When run at the command line with "python top.py --lines=5 alpha.txt beta.txt", the script sets "args.lines" to "5" and "args.filenames" to "['alpha.txt', 'beta.txt']".

10.4. Error Output Redirection and Program Termination
==

The "sys" module also has attributes for *stdin*, *stdout*, and *stderr*. The latter is useful for emitting warnings and error messages to make them visible even when *stdout* has been redirected:

```
>>> sys.stderr.write('Warning, log file not found starting a new one\n')
Warning, log file not found starting a new one
```

The most direct way to terminate a script is to use "sys.exit()".

10.5. String Pattern Matching
==============================

The "re" module provides regular expression tools for advanced string processing. For complex matching and manipulation, regular expressions offer succinct, optimized solutions:

```
>>> import re
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
'cat in the hat'
```

When only simple capabilities are needed, string methods are preferred because they are easier to read and debug:

```
>>> 'tea for too'.replace('too', 'two')
'tea for two'
```

10.6. Mathematics

The "math" module gives access to the underlying C library functions for floating-point math:

```
>>> import math
>>> math.cos(math.pi / 4)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

The "random" module provides tools for making random selections:

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(range(100), 10) # sampling without replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random() # random float from the interval [0.0, 1.0)
0.17970987693706186
>>> random.randrange(6) # random integer chosen from range(6)
4
```

The "statistics" module calculates basic statistical properties (the mean, median, variance, etc.) of numeric data:

```
>>> import statistics
>>> data = [2.75, 1.75, 1.25, 0.25, 0.5, 1.25, 3.5]
>>> statistics.mean(data)
1.6071428571428572
>>> statistics.median(data)
1.25
>>> statistics.variance(data)
1.3720238095238095
```

The SciPy project <https://scipy.org> has many other modules for numerical computations.

10.7. Internet Access

There are a number of modules for accessing the internet and processing internet protocols. Two of the simplest are "urllib.request" for retrieving data from URLs and "smtplib" for sending mail:

```
>>> from urllib.request import urlopen
```

```
>>> with urlopen('https://docs.python.org/3/') as response:
... for line in response:
... line = line.decode() # Convert bytes to a str
... if 'updated' in line:
... print(line.rstrip()) # Remove trailing newline
...
Last updated on Nov 11, 2025 (20:11 UTC).

>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('soothsayer@example.org', 'jcaesar@example.org',
... """To: jcaesar@example.org
... From: soothsayer@example.org
...
... Beware the Ides of March.
... """)
>>> server.quit()
```

(Note that the second example needs a mailserver running on localhost.)

10.8. Dates and Times

The "datetime" module supplies classes for manipulating dates and times in both simple and complex ways. While date and time arithmetic is supported, the focus of the implementation is on efficient member extraction for output formatting and manipulation. The module also supports objects that are timezone aware.

```
>>> # dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

>>> # dates support calendar arithmetic
>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```


10.9. Data Compression

Common data archiving and compression formats are directly supported by modules including: "zlib", "gzip", "bz2", "lzma", "zipfile" and "tarfile".

```
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
```

```
>>> t = zlib.compress(s)
>>> len(t)
37
>>> zlib.decompress(t)
b'witch which has which witches wrist watch'
>>> zlib.crc32(s)
226805979
```

10.10. Performance Measurement

==============================

Some Python users develop a deep interest in knowing the relative
performance of different approaches to the same problem. Python
provides a measurement tool that answers those questions immediately.

For example, it may be tempting to use the tuple packing and unpacking
feature instead of the traditional approach to swapping arguments. The
"timeit" module quickly demonstrates a modest performance advantage:

```
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.57535828626024577
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.54962537085770791
```

In contrast to "timeit"'s fine level of granularity, the "profile" and
"pstats" modules provide tools for identifying time critical sections
in larger blocks of code.

10.11. Quality Control

======================

One approach for developing high quality software is to write tests
for each function as it is developed and to run those tests frequently
during the development process.

The "doctest" module provides a tool for scanning a module and
validating tests embedded in a program's docstrings. Test
construction is as simple as cutting-and-pasting a typical call along
with its results into the docstring. This improves the documentation
by providing the user with an example and it allows the doctest module
to make sure the code remains true to the documentation:

```
def average(values):
    """Computes the arithmetic mean of a list of numbers.

    >>> print(average([20, 30, 70]))
    40.0
    """
    return sum(values) / len(values)

import doctest
doctest.testmod()   # automatically validate the embedded tests
```

The "unittest" module is not as effortless as the "doctest" module, but it allows a more comprehensive set of tests to be maintained in a separate file:

```
import unittest

class TestStatisticalFunctions(unittest.TestCase):

    def test_average(self):
        self.assertEqual(average([20, 30, 70]), 40.0)
        self.assertEqual(round(average([1, 5, 7]), 1), 4.3)
        with self.assertRaises(ZeroDivisionError):
            average([])
        with self.assertRaises(TypeError):
            average(20, 30, 70)

unittest.main() # Calling from the command line invokes all tests
```

10.12. Batteries Included
===========================

Python has a "batteries included" philosophy. This is best seen through the sophisticated and robust capabilities of its larger packages. For example:

* The "xmlrpc.client" and "xmlrpc.server" modules make implementing remote procedure calls into an almost trivial task. Despite the modules' names, no direct knowledge or handling of XML is needed.

* The "email" package is a library for managing email messages, including MIME and other **RFC 5322**-based message documents. Unlike "smtplib" and "poplib" which actually send and receive messages, the email package has a complete toolset for building or decoding complex message structures (including attachments) and for implementing internet encoding and header protocols.

* The "json" package provides robust support for parsing this popular data interchange format. The "csv" module supports direct reading and writing of files in Comma-Separated Value format, commonly supported by databases and spreadsheets. XML processing is supported by the "xml.etree.ElementTree", "xml.dom" and "xml.sax" packages. Together, these modules and packages greatly simplify data interchange between Python applications and other tools.

* The "sqlite3" module is a wrapper for the SQLite database library, providing a persistent database that can be updated and accessed using slightly nonstandard SQL syntax.

* Internationalization is supported by a number of modules including "gettext", "locale", and the "codecs" package.