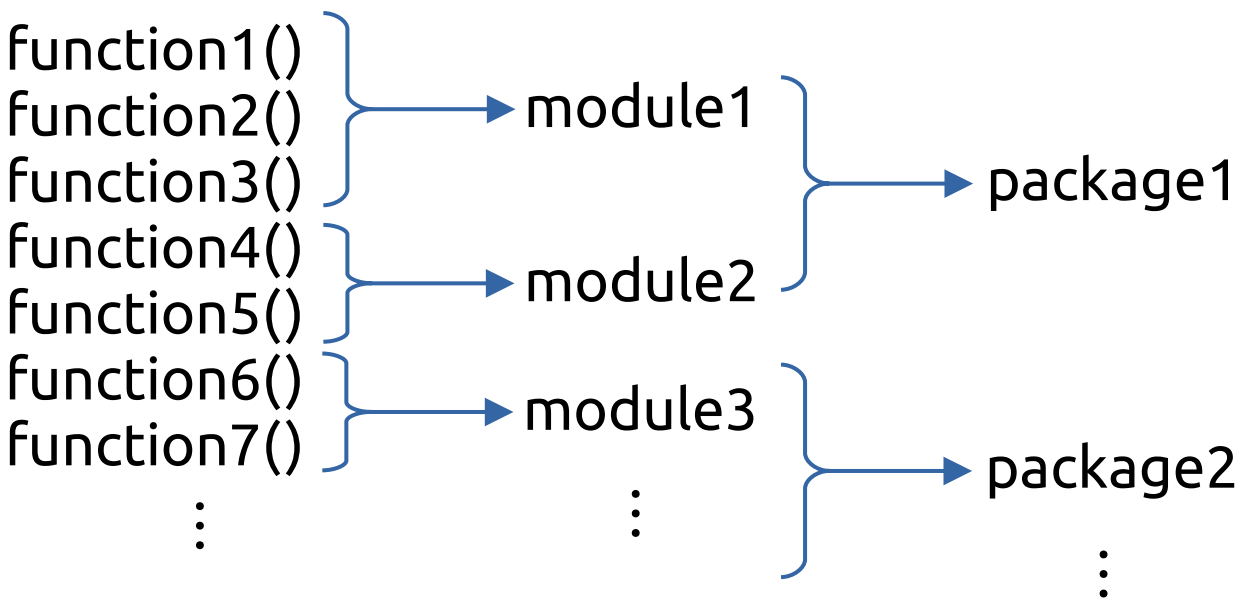
Module 3
Functions, Modules, and Packages / The Standard Library

Lots of code:


function1()
function2()
function3()
function4()
function5()
function6()
function7()
module1
module2
module3
package1
package2

Defining a function

Syntax:

```
def <name>(arg1,…,argN,dftarg1=<value>,…,dftargM=<value>):
    """ <docstring> """
   <body>
   return <output>
```

Notes:

- Use “pass” for a function with no body.
- Some arguments may define default values, but they must be listed last.
- The arguments, the docstring, and the return value are optional.

Calling a function

Notes:

- Can omit arguments with defined default values.
- Two style of passing arguments: **positional** and **keyword**

Example:

```
def myfunc(a,b,c=0,d=1):
    print(a,b,c,d)
```

```
myfunc(1,2,3,4)      → 1 2 3 4      … positional only
myfunc(1,2,3)        → 1 2 3 1      … positional only w/ defaults
myfunc(a=4,b=5)      → 4 5 0 1      … keyword only w/ defaults
myfunc(4,b=5,d=8)    → 4 5 0 8      … positional & keyword
myfunc(a=4,5,d=8)    → SyntaxError
```

Return values

- Python functions return a single value.
- We can emulate multiple return values by:
  - in the definition: return a parentheses-less tuple,
  - in the call: unpack the result.
- Keyword style: return a dictionary.

Modules

- A **module** is a **`.py` file** with functions, variables, classes, etc.
- Use it to store items (e.g. utilities) that you want to use in >1 script.
- Use the **import** statement to load a module or a portion thereof.
- **import** statements are usually placed at the top of the file.

moduleA.py

```
def sayHi():
    print("Hello from A!")

def sayA():
    print("A")
```

Try it:

```
>> import moduleA
>> sayHi()              # ERROR
>> moduleA.sayHi()
```

namespace
for moduleA

“from”

- Use **from** to load specific parts of a module.

```
>> from moduleA import sayA
>> moduleA.sayHi()   # ERROR
>> moduleA.sayA()    # ERROR
>> sayA()            # ‘A’
```

Shadowing (namespace clash)

moduleA.py

```
def sayHi():
    print("Hello from A!")

def sayA():
    print("A")
```

moduleB.py

```
def sayHi():
    print("Hello from B!")

def sayB():
    print("B")
```

```
>> from moduleA import *
>> from moduleB import *
>> sayHi()  # Hello from B!
```

“as”

- Assign aliases to imported items

```
>> import moduleA as mA
>> from moduleB import sayHi as BsayHi
>> mA.sayHi()         # Hello from A!
>> BsayHi()           # Hello from B!
```

Packages

- A **package** is a set of modules that work together.
- A package is (usually) contained in a single folder. (e.g. `mypackage/`)
- A folder is designated as a package by adding an `__init__.py file.`
- `__init__.py` contains package-level information.
- Packages can be “packaged” and published on PyPI. You can then install them with pip.
- Python also searches for packages on your system path.

Previous topic
10. Full Grammar specification

Next topic
Introduction

This Page
Report a Bug
Show Source

The Python Standard Library

While The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is an active collection of hundreds of thousands of components (from individual programs and modules to packages and entire application development frameworks), available from the Python Package Index.

https://docs.python.org/3/library/index.html

Built-in functions

Built-in Functions			
A abs() aiter() all() anext() any() ascii() **B** bin() bool() breakpoint() bytearray() bytes() **C** callable() chr() classmethod() compile() complex() **D** delattr() dict() dir() divmod()	**E** enumerate() eval() exec() **F** filter() float() format() frozenset() **G** getattr() globals() **H** hasattr() hash() help() hex() **I** id() input() int() isinstance() issubclass() iter()	**L** len() list() locals() **M** map() max() memoryview() min() **N** next() **O** object() oct() open() ord() **P** pow() print() property()	**R** range() repr() reversed() round() **S** set() setattr() slice() sorted() staticmethod() str() sum() super() **T** tuple() type() **V** vars() **Z** zip() _ __import__()

Built-in Functions

A
abs()
aiter()
all()
any()
anext()
ascii()

B
bin()
bool()
breakpoint()
bytearray()
bytes()

C
callable()
chr()
classmethod()
compile()
complex()

D
delattr()
dict()
dir()
divmod()

E
enumerate()
eval()
exec()

F
filter()
float()
format()
frozenset()

G
getattr()
globals()

H
hasattr()
hash()
help()
hex()

I
id()
input()
int()
isinstance()
issubclass()
iter()

L
len()
list()
locals()

M
map()
max()
memoryview()
min()

N
next()

O
object()
oct()
open()
ord()

P
pow()
print()
property()

R
range()
repr()
reversed()
round()

S
set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T
tuple()
type()

V
vars()

Z
zip()

_
__import__()

type constructors

- int()
- float()
- bool()
- str()
- tuple()
- list()
- set()
- dict()

used in iteration

- range()
- enumerate()

type checking

- type()
- isinstance()

iterable summaries

- len()
- all()
- any()
- max()
- min()
- sum()

input/output

- input()
- print()
- open()

Standard library packages

- Text Processing Services
  - `string` — Common string operations
  - `re` — Regular expression operations
  - `difflib` — Helpers for computing deltas
  - `textwrap` — Text wrapping and filling
  - `unicodedata` — Unicode Database
  - `stringprep` — Internet String Preparation
  - `readline` — GNU readline interface
  - `rlcompleter` — Completion function for GNU readline
- Binary Data Services
  - `struct` — Interpret bytes as packed binary data
  - `codecs` — Codec registry and base classes
- Data Types
  - `datetime` — Basic date and time types
  - `zoneinfo` — IANA time zone support
  - `calendar` — General calendar-related functions
  - `collections` — Container datatypes
  - `collections.abc` — Abstract Base Classes for Containers
  - `heapq` — Heap queue algorithm
  - `bisect` — Array bisection algorithm
  - `array` — Efficient arrays of numeric values
  - `weakref` — Weak references
  - `types` — Dynamic type creation and names for built-in types
  - `copy` — Shallow and deep copy operations
  - `pprint` — Data pretty printer
  - `reprlib` — Alternate `repr()` implementation
  - `enum` — Support for enumerations
  - `graphlib` — Functionality to operate with graph-like structures
- 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
  - `operator` — Standard operators as functions
- File and Directory Access
  - `pathlib` — Object-oriented filesystem paths
  - `os.path` — Common pathname manipulations
  - `fileinput` — Iterate over lines from multiple input streams
  - `stat` — Interpreting `stat()` results
  - `filecmp` — File and Directory Comparisons
  - `tempfile` — Generate temporary files and directories
  - `glob` — Unix style pathname pattern expansion
  - `fnmatch` — Unix filename pattern matching
  - `linecache` — Random access to text lines
  - `shutil` — High-level file operations
- Data Persistence
  - `pickle` — Python object serialization
  - `copyreg` — Register `pickle` support functions
  - `shelve` — Python object persistence
  - `marshal` — Internal Python object serialization
  - `dbm` — Interfaces to Unix "databases"
  - `sqlite3` — DB-API 2.0 interface for SQLite databases
- Data Compression and Archiving
  - `zlib` — Compression compatible with **gzip**
  - `gzip` — Support for **gzip** files
  - `bz2` — Support for **bzip2** compression
  - `lzma` — Compression using the LZMA algorithm
  - `zipfile` — Work with ZIP archives
  - `tarfile` — Read and write tar archive files
- File Formats
  - `csv` — CSV File Reading and Writing
  - `configparser` — Configuration file parser
  - `tomllib` — Parse TOML files
  - `netrc` — netrc file processing
  - `plistlib` — Generate and parse Apple `.plist` files
- Cryptographic Services
  - `hashlib` — Secure hashes and message digests
  - `hmac` — Keyed-Hashing for Message Authentication
  - `secrets` — Generate secure random numbers for managing secrets
- Generic Operating System Services
  - `os` — Miscellaneous operating system interfaces
  - `io` — Core tools for working with streams
  - `time` — Time access and conversions
  - `argparse` — Parser for command-line options, arguments and sub-c
  - `getopt` — C-style parser for command line options
  - `logging` — Logging facility for Python
  - `logging.config` — Logging configuration
  - `logging.handlers` — Logging handlers
  - `getpass` — Portable password input
  - `curses` — Terminal handling for character-cell displays
  - `curses.textpad` — Text input widget for curses programs
  - `curses.ascii` — Utilities for ASCII characters
  - `curses.panel` — A panel stack extension for curses
  - `platform` — Access to underlying platform's identifying data
  - `errno` — Standard errno system symbols
  - `ctypes` — A foreign function library for Python
- Concurrent Execution
  - `threading` — Thread-based parallelism
  - `multiprocessing` — Process-based parallelism
  - `multiprocessing.shared_memory` — Shared memory for direct a
  - The `concurrent` package
  - `concurrent.futures` — Launching parallel tasks
  - `subprocess` — Subprocess management
  - `sched` — Event scheduler
  - `queue` — A synchronized queue class
  - `contextvars` — Context Variables
  - `_thread` — Low-level threading API
- Networking and Interprocess Communication
  - `asyncio` — Asynchronous I/O
  - `socket` — Low-level networking interface
  - `ssl` — TLS/SSL wrapper for socket objects
  - `select` — Waiting for I/O completion
  - `selectors` — High-level I/O multiplexing
  - `signal` — Set handlers for asynchronous events
  - `mmap` — Memory-mapped file support
- Internet Data Handling
  - `email` — An email and MIME handling package
  - `json` — JSON encoder and decoder
  - `mailbox` — Manipulate mailboxes in various formats
  - `mimetypes` — Map filenames to MIME types
  - `base64` — Base16, Base32, Base64, Base85 Data Encodings
  - `binascii` — Convert between binary and ASCII
  - `quopri` — Encode and decode MIME quoted-printable data
- Structured Markup Processing Tools
  - `html` — HyperText Markup Language support
  - `html.parser` — Simple HTML and XHTML parser
  - `html.entities` — Definitions of HTML general entities
  - XML Processing Modules
  - `xml.etree.ElementTree` — The ElementTree XML API
  - `xml.dom` — The Document Object Model API
  - `xml.dom.minidom` — Minimal DOM implementation
  - `xml.dom.pulldom` — Support for building partial DOM trees
  - `xml.sax` — Support for SAX2 parsers
  - `xml.sax.handler` — Base classes for SAX handlers
  - `xml.sax.saxutils` — SAX Utilities
  - `xml.sax.xmlreader` — Interface for XML parsers
  - `xml.parsers.expat` — Fast XML parsing using Expat
- Internet Protocols and Support
  - `webbrowser` — Convenient web-browser controller
  - `wsgiref` — WSGI Utilities and Reference Implementation
  - `urllib` — URL handling modules
  - `urllib.request` — Extensible library for opening URLs
  - `urllib.response` — Response classes used by urllib
  - `urllib.parse` — Parse URLs into components
  - `urllib.error` — Exception classes raised by urllib.request
  - `urllib.robotparser` — Parser for robots.txt
  - `http` — HTTP modules
  - `http.client` — HTTP protocol client
  - `ftplib` — FTP protocol client
  - `poplib` — POP3 protocol client
  - `imaplib` — IMAP4 protocol client
  - `smtplib` — SMTP protocol client
  - `uuid` — UUID objects according to **RFC 4122**
  - `socketserver` — A framework for network servers
  - `http.server` — HTTP servers
  - `http.cookies` — HTTP state management
  - `http.cookiejar` — Cookie handling for HTTP clients
  - `xmlrpc` — XMLRPC server and client modules
  - `xmlrpc.client` — XML-RPC client access
  - `xmlrpc.server` — Basic XML-RPC servers
  - `ipaddress` — IPv4/IPv6 manipulation library
- Multimedia Services
  - `wave` — Read and write WAV files
  - `colorsys` — Conversions between color systems
- Internationalization
  - `gettext` — Multilingual internationalization services
  - `locale` — Internationalization services
- Program Frameworks
  - `turtle` — Turtle graphics
  - `cmd` — Support for line-oriented command interpreters
  - `shlex` — Simple lexical analysis
- Graphical User Interfaces with Tk
  - `tkinter` — Python interface to Tcl/Tk
  - `tkinter.colorchooser` — Color choosing dialog
  - `tkinter.font` — Tkinter font wrapper
  - Tkinter Dialogs
  - `tkinter.messagebox` — Tkinter message prompts
  - `tkinter.scrolledtext` — Scrolled Text Widget
  - `tkinter.dnd` — Drag and drop support
  - `tkinter.ttk` — Tk themed widgets
  - `tkinter.tix` — Extension widgets for Tk
  - IDLE
- Development Tools
  - `typing` — Support for type hints
  - `pydoc` — Documentation generator and online help system
  - Python Development Mode
  - `doctest` — Test interactive Python examples
  - `unittest` — Unit testing framework
  - `unittest.mock` — mock object library
  - `unittest.mock` — getting started
  - 2to3 — Automated Python 2 to 3 code translation
  - `test` — Regression tests package for Python
  - `test.support` — Utilities for the Python test suite
  - `test.support.socket_helper` — Utilities for socket tests
  - `test.support.script_helper` — Utilities for the Python executic
  - `test.support.bytecode_helper` — Support tools for testing cor
  - `test.support.threading_helper` — Utilities for threading tests
  - `test.support.os_helper` — Utilities for os tests
  - `test.support.import_helper` — Utilities for import tests
  - `test.support.warnings_helper` — Utilities for warnings tests
- Debugging and Profiling
  - Audit events table
  - `bdb` — Debugger framework
  - `faulthandler` — Dump the Python traceback
  - `pdb` — The Python Debugger
  - The Python Profilers
  - `timeit` — Measure execution time of small code snippets
  - `trace` — Trace or track Python statement execution
  - `tracemalloc` — Trace memory allocations
- Software Packaging and Distribution
  - `distutils` — Building and installing Python modules
  - `ensurepip` — Bootstrapping the `pip` installer
  - `venv` — Creation of virtual environments
  - `zipapp` — Manage executable Python zip archives
- Python Runtime Services
  - `sys` — System-specific parameters and functions
  - `sysconfig` — Provide access to Python's configuration information
  - `builtins` — Built-in objects
  - `__main__` — Top-level code environment
  - `warnings` — Warning control
  - `dataclasses` — Data Classes
  - `contextlib` — Utilities for `with`-statement contexts
  - `abc` — Abstract Base Classes
  - `atexit` — Exit handlers
  - `traceback` — Print or retrieve a stack traceback
  - `__future__` — Future statement definitions
  - `gc` — Garbage Collector interface
  - `inspect` — Inspect live objects
  - `site` — Site-specific configuration hook
- Custom Python Interpreters
  - `code` — Interpreter base classes
  - `codeop` — Compile Python code
- Importing Modules
  - `zipimport` — Import modules from Zip archives
  - `pkgutil` — Package extension utility
  - `modulefinder` — Find modules used by a script
  - `runpy` — Locating and executing Python modules
  - `importlib` — The implementation of `import`
  - `importlib.resources` – Package resource reading, opening and acc
  - `importlib.resources.abc` – Abstract base classes for resources
  - `importlib.metadata` – Accessing package metadata
  - The initialization of the `sys.path` module search path
- Python Language Services
  - `ast` — Abstract Syntax Trees
  - `symtable` — Access to the compiler's symbol tables
  - `token` — Constants used with Python parse trees
  - `keyword` — Testing for Python keywords
  - `tokenize` — Tokenizer for Python source
  - `tabnanny` — Detection of ambiguous indentation
  - `pyclbr` — Python module browser support
  - `py_compile` — Compile Python source files
  - `compileall` — Byte-compile Python libraries
  - `dis` — Disassembler for Python bytecode
  - `pickletools` — Tools for pickle developers
- MS Windows Specific Services
  - `msvcrt` — Useful routines from the MS VC++ runtime
  - `winreg` — Windows registry access
  - `winsound` — Sound-playing interface for Windows
- Unix Specific Services
  - `posix` — The most common POSIX system calls
  - `pwd` — The password database
  - `grp` — The group database
  - `termios` — POSIX style tty control
  - `tty` — Terminal control functions
  - `pty` — Pseudo-terminal utilities
  - `fcntl` — The `fcntl` and `ioctl` system calls
  - `resource` — Resource usage information
  - `syslog` — Unix syslog library routines

math module

Constants

```
pi   inf
e    nan
```

Comparisons

```
isclose(a,b,tol)
isfinite(x)
isinf(x)
isnan(x)
```

Rounding

```
fabs(x)
ceil(x)
modf(x)
floor(x)
trunc(x)
```

Modular Arithmetic

```
fmod(x, y)
remainder(x, y)
gcd(*integers)
lcm(*integers)
```

Comninatorics

```
comb(n, k)
perm(n, k)
factorial(n)
```

Trigonometry

```
cos(x)      acos(x)
sin(x)      asin(x)
tan(x)      atan(x)
radians(x)  atan2(x,y)
degrees(x)
```

Exponential functions

```
exp(x)        pow(x, y)
exp2(x)       sqrt(x)
log(x,base)   cbrt(x)
log2(x)
log10(x)
```

random module

Random seed

```
seed(a)
```

Sample integers

```
randint(a, b)
```

Sample sequences

```
choice(seq)
sample(seq,k)
shuffle(seq)
```

Sampling real numbers

```
random()
uniform(a, b)
gauss(mu,sigma)
triangular(low, high, mode)
betavariate(alpha, beta)
    ⋮
```

Working with files

Operating system modules

- `os` : make/remove files/folders, environment variables, etc.
- `os.path` : Absolute and relative file/folder names.

File I/O modules

- `csv` : Tabular data in a text format.
- `pickle` : Binary format for Python objects.

UNIX terminology

- **file** : document
- **directory** : folder (a container of files)
- **path/pathname/filename :** A string that locates the file or directory in the file system
    - **absolute** : with respect to the root of the file system.
    - **relative** : with respect to another directory.
        - . means “this directory”
        - .. means “my parent directory”

- **terminal / command-line interface (CLI)** : A program used to send commands and receive responses with the operating system. (e.g. **bash**)
- **current directory** : directory currently referred to by the CLI.

os module

Function name	What it does
`getcwd()`	Returns the name of the current working directory
`chdir(str)`	Switches to a new current working directory
`listdir()`	Lists the contents of the current working directory
`makedirs(str)`	Make new directory (possible many nested)
`rmdir(str)`	Remove a directory
`rename(old,new)`	Rename a file or directory

`os.path` module

Function name	Returns
`isabs(path)`	Is path an absolute path?
`exists(path)`	Is path an existing file or directory?
`isdir(path)`	Is path an existing directory?
`isfile(path)`	Is path an existing file?

`os.path` module

extension

`/home/gomes/235A_p4e/dist/module2/2_2_iterables.py`

dirname

basename

	Function name	Returns
deconstruct	`dirname(path)`	dirname
	`basename(path)`	basename
	`split(path)`	(dirname,basename)
	`splitext(path)`	(dirname/basename,extension)
build	`join(*paths)`	Concatenate paths (all except last must be directories)
	`abspath(path)`	Normalize and absolutize a path
	`relpath(path)`	Normalize and relativize a path

Example

create a path string

```
home = os.environ.get("HOME")
basename = "tempfile.txt"
filepath = os.path.join(home,basename)
```

open the file
do something with it
close the file

```
f = open(filepath,"w")
f.write("hello!")
f.close()
```

… {"r","w","+","a","b","t"}
… read,readline,write,etc.

or

use a context manager
(`with` keyword)

```
with open(filepath,"w") as f:
    f.write("hello!")
```

csv module

import the module

```
import csv
```

write to a file

```
with open(outfile,'w') as f
    csv_writer = csv.writer(f)
    for line in A:
        csv_writer.write(line)
```

read from a file

```
with open(infile,'r') as f
    csv_reader = csv.reader(f)
    next(csv_reader)  # skip the header
    for line in csv_reader:
        print(line)
```

`pickle` module

import the module

```
import pickle
```

write to a file

```
with open(outfile,'wb') as f
    pickle.dump(A,f)
```

read from a file

```
with open(infile,'rb') as f
    A = pickle.load(f)
```