```
(base) PS C:\Users\Abi Rahman> python
Python 3.10.9 | packaged by conda-forge | (main, Jan 11 2023, 15:15:40) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> #10. Brief Tour of the Standard Library
>>> #10.1. Operating System Interface
>>> import os
>>> os.getcwd()
'C:\\Users\\Abi Rahman'
>>> os.system('mkdir today')   # Run the command mkdir in the system shell
0
>>>
>>> import os
>>> dir(os)
['DirEntry', 'F_OK', 'GenericAlias', 'Mapping', 'MutableMapping', 'O_APPEND', 'O_BINARY',
'O_CREAT', 'O_EXCL', 'O_NOINHERIT', 'O_RANDOM', 'O_RDONLY', 'O_RDWR', 'O_SEQUENTIAL',
'O_SHORT_LIVED', 'O_TEMPORARY', 'O_TEXT', 'O_TRUNC', 'O_WRONLY', 'P_DETACH', 'P_NOWAIT',
'P_NOWAITO', 'P_OVERLAY', 'P_WAIT', 'PathLike', 'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET',
'TMP_MAX', 'W_OK', 'X_OK', '_AddedDllDirectory', '_Environ', '__all__', '__builtins__', '__cached__',
'__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '_check_methods',
'_execvpe', '_exists', '_exit', '_fspath', '_get_exports_list', '_walk', '_wrap_close', 'abc', 'abort',
'access', 'add_dll_directory', 'altsep', 'chdir', 'chmod', 'close', 'closerange', 'cpu_count', 'curdir',
'defpath', 'device_encoding', 'devnull', 'dup', 'dup2', 'environ', 'error', 'execl', 'execle', 'execlp',
'execlpe', 'execv', 'execve', 'execvp', 'execvpe', 'extsep', 'fdopen', 'fsdecode', 'fsencode', 'fspath',
'fstat', 'fsync', 'ftruncate', 'get_exec_path', 'get_handle_inheritable', 'get_inheritable',
'get_terminal_size', 'getcwd', 'getcwdb', 'getenv', 'getlogin', 'getpid', 'getppid', 'isatty', 'kill', 'linesep',
'link', 'listdir', 'lseek', 'lstat', 'makedirs', 'mkdir', 'name', 'open', 'pardir', 'path', 'pathsep', 'pipe',
'popen', 'putenv', 'read', 'readlink', 'remove', 'removedirs', 'rename', 'renames', 'replace', 'rmdir',
'scandir', 'sep', 'set_handle_inheritable', 'set_inheritable', 'spawnl', 'spawnle', 'spawnv', 'spawnve',
'st', 'startfile', 'stat', 'stat_result', 'statvfs_result', 'strerror', 'supports_bytes_environ',
'supports_dir_fd', 'supports_effective_ids', 'supports_fd', 'supports_follow_symlinks', 'symlink', 'sys',
'system', 'terminal_size', 'times', 'times_result', 'truncate', 'umask', 'uname_result', 'unlink',
'unsetenv', 'urandom', 'utime', 'waitpid', 'waitstatus_to_exitcode', 'walk', 'write']
>>> help(os)
Help on module os:

NAME
    os - OS routines for NT or Posix depending on what system we're on.
```

MODULE REFERENCE

https://docs.python.org/3.10/library/os.html

The following documentation is automatically generated from the Python source files. It may be incomplete, incorrect or include features that are considered implementation detail and may vary between Python implementations. When in doubt, consult the module reference at the location listed above.

DESCRIPTION

This exports:

- all functions from posix or nt, e.g. unlink, stat, etc.
- os.path is either posixpath or ntpath
- os.name is either 'posix' or 'nt'
- os.curdir is a string representing the current directory (always '.')
- os.pardir is a string representing the parent directory (always '..')
- os.sep is the (or a most common) pathname separator ('/' or '\\')
- os.extsep is the extension separator (always '.')
- os.altsep is the alternate pathname separator (None or '/')
- os.pathsep is the component separator used in $PATH etc
- os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
- os.defpath is the default search path for executables
- os.devnull is the file path of the null device ('/dev/null', etc.)

Programs that import and use 'os' stand a better chance of being portable between different platforms. Of course, they must then only use functions that are defined by all platforms (e.g., unlink and opendir), and leave all pathname manipulation to os.path (e.g., split and join).

```
CLASSES
    builtins.Exception(builtins.BaseException)
        builtins.OSError
    builtins.object
        nt.DirEntry
    builtins.tuple(builtins.object)
        nt.times_result
-- More  --
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\Abi Rahman\miniconda3\lib\shutil.py", line 254, in copyfile
    with open(src, 'rb') as fsrc:
FileNotFoundError: [Errno 2] No such file or directory: 'data.db'
>>> shutil.move('/build/executables', 'installdir')
Traceback (most recent call last):
  File "C:\Users\Abi Rahman\miniconda3\lib\shutil.py", line 816, in move
    os.rename(src, real_dst)
FileNotFoundError: [WinError 3] The system cannot find the path specified: '/build/executables' ->
'installdir'


During handling of the above exception, another exception occurred:


Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\Abi Rahman\miniconda3\lib\shutil.py", line 836, in move
    copy_function(src, real_dst)
  File "C:\Users\Abi Rahman\miniconda3\lib\shutil.py", line 434, in copy2
    copyfile(src, dst, follow_symlinks=follow_symlinks)
  File "C:\Users\Abi Rahman\miniconda3\lib\shutil.py", line 254, in copyfile
```

```
   with open(src, 'rb') as fsrc:
FileNotFoundError: [Errno 2] No such file or directory: '/build/executables'
>>>



>>>
>>> #10.2. File Wildcards
>>> import glob
>>> glob.glob('*.py')
[]
>>> #10.3. Command Line Arguments
>>> import sys
>>> print(sys.argv)
['']

>>> #10.5. String Pattern Matching
>>> 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'
>>>
>>> 'tea for too'.replace('too', 'two')
'tea for two'
>>>
>>>
>>> #10.6. Mathematics
>>> import math
>>> math.cos(math.pi / 4)
0.7071067811865476
>>> math.log(1024, 2)
```

```
10.0
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'banana'
>>> random.sample(range(100), 10)   # sampling without replacement
[23, 48, 63, 26, 93, 40, 32, 6, 58, 42]
>>> random.random()    # random float
0.8433650672983969
>>> random.randrange(6)    # random integer chosen from range(6)
3
>>> 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
>>>
>>>
>>> #10.7. Internet Access
>>> from urllib.request import urlopen
>>> with urlopen('http://worldtimeapi.org/api/timezone/etc/UTC.txt') as response:
...     for line in response:
...         line = line.decode()             # Convert bytes to a str
...         if line.startswith('datetime'):
...             print(line.rstrip())
...
datetime: 2023-08-25T06:42:17.777870+00:00
>>> server.sendmail('soothsayer@example.org', 'jcaesar@example.org',
...     """To: jcaesar@example.org
```

```
...     From: soothsayer@example.org

...
...     Beware the Ides of March.
...     """)
>>>server.quit()
>>>
>>>
>>> #10.8. Dates and Times
>>> # dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2023, 8, 25)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'08-25-23. 25 Aug 2023 is a Friday on the 25 day of August.'
>>>
>>> # dates support calendar arithmetic
>>> birthday = date(2001, 3, 16)
>>> age = now - birthday
>>> age.days
8197
>>>
>>>
>>> #10.9. Data Compression
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
41
>>> 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
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.06509689999802504
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.07148859999142587
>>>
>>>
>>> #10.11. Quality Control
>>> 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
TestResults(failed=0, attempted=1)
>>>
>>> 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()
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(base) PS C:\Users\Abi Rahman>
```

```
(base) PS C:\Users\Abi Rahman> python
Python 3.10.9 | packaged by conda-forge | (main, Jan 11 2023, 15:15:40) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> #11. Brief Tour of the Standard Library - Part II
>>> #11.1. Output Formatting
>>> import reprlib
>>> reprlib.repr(set('supercalifragilisticexpialidocious'))
"{'a', 'c', 'd', 'e', 'f', 'g', ...}"
>>>
>>> import pprint
>>> t = [[[['black', 'cyan'], 'white', ['green', 'red']], [['magenta',
...     'yellow'], 'blue']]]
>>> pprint.pprint(t, width=30)
[[[['black', 'cyan'],
   'white',
   ['green', 'red']],
  [['magenta', 'yellow'],
   'blue']]]
>>>
>>> import textwrap
>>> doc = """The wrap() method is just like fill() except that it returns
... a list of strings instead of one big string with newlines to separate
... the wrapped lines."""
>>> print(textwrap.fill(doc, width=40))
The wrap() method is just like fill()
except that it returns a list of strings
instead of one big string with newlines
to separate the wrapped lines.
>>>
>>> import locale
```

```
>>> locale.setlocale(locale.LC_ALL, 'English_United States.1252')
'English_United States.1252'
>>> conv = locale.localeconv()          # get a mapping of conventions
>>> x = 1234567.8
>>> locale.format("%d", x, grouping=True)
<stdin>:1: DeprecationWarning: This method will be removed in a future version of Python. Use
'locale.format_string()' instead.
'1,234,567'
>>> locale.format_string("%s%.*f", (conv['currency_symbol'],
...                      conv['frac_digits'], x), grouping=True)
'$1,234,567.80'
>>>
>>>
>>> #11.2. Templating
>>> from string import Template
>>> t = Template('${village}folk send $$10 to $cause.')
>>> t.substitute(village='Nottingham', cause='the ditch fund')
'Nottinghamfolk send $10 to the ditch fund.'
>>> t = Template('Return the $item to $owner.')
>>> d = dict(item='unladen swallow')
>>> t.substitute(d)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\Abi Rahman\miniconda3\lib\string.py", line 121, in substitute
    return self.pattern.sub(convert, self.template)
  File "C:\Users\Abi Rahman\miniconda3\lib\string.py", line 114, in convert
    return str(mapping[named])
KeyError: 'owner'
>>> t.safe_substitute(d)
'Return the unladen swallow to $owner.'
>>>
```

```
>>> import time, os.path
>>> photofiles = ['img_1074.jpg', 'img_1076.jpg', 'img_1077.jpg']
>>> class BatchRename(Template):
...     delimiter = '%'
...
>>> fmt = input('Enter rename style (%d-date %n-seqnum %f-format):  ')
Enter rename style (%d-date %n-seqnum %f-format):  Ashley_%n%f
>>>
>>> t = BatchRename(fmt)
>>> date = time.strftime('%d%b%y')
>>> for i, filename in enumerate(photofiles):
...     base, ext = os.path.splitext(filename)
...     newname = t.substitute(d=date, n=i, f=ext)
...     print('{0} --> {1}'.format(filename, newname))
...
img_1074.jpg --> Ashley_0.jpg
img_1076.jpg --> Ashley_1.jpg
img_1077.jpg --> Ashley_2.jpg
>>>
>>>
>>> #11.4. Multi-threading
>>> import threading, zipfile
>>> class AsyncZip(threading.Thread):
...     def __init__(self, infile, outfile):
...         threading.Thread.__init__(self)
...         self.infile = infile
...         self.outfile = outfile
...     def run(self):
...         f = zipfile.ZipFile(self.outfile, 'w', zipfile.ZIP_DEFLATED)
...         f.write(self.infile)
...         f.close()
```

```
...         print('Finished background zip of:', self.infile)
...
>>> background = AsyncZip('mydata.txt', 'myarchive.zip')
>>> background.start()
>>> Exception in thread Thread-1:
Traceback (most recent call last):
  File "C:\Users\Abi Rahman\miniconda3\lib\threading.py", line 1016, in _bootstrap_inner
    self.run()
  File "<stdin>", line 8, in run
  File "C:\Users\Abi Rahman\miniconda3\lib\zipfile.py", line 1739, in write
    zinfo = ZipInfo.from_file(filename, arcname,
  File "C:\Users\Abi Rahman\miniconda3\lib\zipfile.py", line 502, in from_file
    st = os.stat(filename)
FileNotFoundError: [WinError 2] The system cannot find the file specified: 'mydata.txt'

>>> print('The main program continues to run in foreground.')
The main program continues to run in foreground.
>>> background.join()    # Wait for the background task to finish
>>> print('Main program waited until background was done.')
Main program waited until background was done.
>>>
>>>
>>> #11.5. Logging
>>> import logging
>>> logging.debug('Debugging information')
>>> logging.info('Informational message')
>>> logging.warning('Warning:config file %s not found', 'server.conf')
WARNING:root:Warning:config file server.conf not found
>>> logging.error('Error occurred')
ERROR:root:Error occurred
>>> logging.critical('Critical error -- shutting down')
```

```
CRITICAL:root:Critical error -- shutting down
>>>
>>>
>>> #11.6. Weak References
>>> import weakref, gc
>>> class A:
...     def __init__(self, value):
...         self.value = value
...     def __repr__(self):
...         return str(self.value)
...
>>> a = A(10)                  # create a reference
>>> d = weakref.WeakValueDictionary()
>>> d['primary'] = a           # does not create a reference
>>> d['primary']               # fetch the object if it is still alive
10
>>> del a                      # remove the one reference
>>> gc.collect()               # run garbage collection right away
0
>>> d['primary']               # entry was automatically removed
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\Abi Rahman\miniconda3\lib\weakref.py", line 137, in __getitem__
    o = self.data[key]()
KeyError: 'primary'
>>>
>>> #
>>> #11.7. Tools for Working with Lists
>>> from array import array
>>> a = array('H', [4000, 10, 700, 22222])
>>> sum(a)
```

```
26932
>>> a[1:3]
array('H', [10, 700])
>>> unsearched = deque([starting_node])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'deque' is not defined
>>> def breadth_first_search(unsearched):
...     node = unsearched.popleft()
...     for m in gen_moves(node):
...         if is_goal(m):
...             return m
...         unsearched.append(m)
...
>>>
>>> import bisect
>>> scores = [(100, 'perl'), (200, 'tcl'), (400, 'lua'), (500, 'python')]
>>> bisect.insort(scores, (300, 'ruby'))
>>> scores
[(100, 'perl'), (200, 'tcl'), (300, 'ruby'), (400, 'lua'), (500, 'python')]
>>>
>>>
>>> from heapq import heapify, heappop, heappush
>>> data = [1, 3, 5, 7, 9, 2, 4, 6, 8, 0]
>>> heapify(data)                      # rearrange the list into heap order
>>> heappush(data, -5)                 # add a new entry
>>> [heappop(data) for i in range(3)]  # fetch the three smallest entries
[-5, 0, 1]
>>>
>>>
>>> #11.8. Decimal Floating Point Arithmeti
```

```
>>> from decimal import *
>>> round(Decimal('0.70') * Decimal('1.05'), 2)
Decimal('0.74')
>>> round(.70 * 1.05, 2)
0.73
>>> Decimal('1.00') % Decimal('.10')
Decimal('0.00')
>>> 1.00 % 0.10
0.09999999999999995
>>> sum([Decimal('0.1')]*10) == Decimal('1.0')
True
>>> sum([0.1]*10) == 1.0
False
>>> getcontext().prec = 36
>>> Decimal(1) / Decimal(7)
Decimal('0.142857142857142857142857142857142857')
>>>
```