CORE PYTHON

AT

ACADEMY OF COMPUTER STUDIES

A Practical Training Report

Submitted in partial fulfillment of the requirement in partial fulfilmet of the requirement for the award of the

DIPLOMA IN INFORMATION TECHNOLOGY

FROM

GOVERNMENT POLYTECHNIC NARANDERA NAGER(T.G)

SUBMITTED BY:- SUBMITTED TO:-

PULKIT SAINI MR. PIYUSH KALA

HOD IT DEPARTMENT

ACKNOWLEDGMENT

On the completion of my traning. I would like to express my sincere thanks to all the those who have guided, advised, inspired & supported me during my training at ”**ACADEMY OF COMPUTER STUDIES**” has completed 45 days of training computer learning & training course entitled “**CORE PYTHON**”.

I would extend my sincere thanks to **Mr. Sunny Singh** for all academic facilities and practical for the completion of training.

I am greatly in debted to everyone who has helped me in one way or other towards the completion of training.

Above all I must and do thank god almight from the depth of my heart for being with me giving hope, confidence and courage to get the task completion in time

**The Python Tutorial**

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see [The Python Standard Library](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/index.html#library-index). [The Python Language Reference](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/index.html#reference-index) gives a more formal definition of the language. To write extensions in C or C++, read [Extending and Embedding the Python Interpreter](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/extending/index.html#extending-index) and [Python/C API Reference Manual](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/c-api/index.html#c-api-index). There are also several books covering Python in depth.

This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Python’s most noteworthy features, and will give you a good idea of the language’s flavor and style. After reading it, you will be able to read and write Python modules and programs, and you will be ready to learn more about the various Python library modules described in [The Python Standard Library](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/index.html#library-index)

## History of the software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see <https://www.cwi.nl/>) in the Netherlands as a successor of a language called ABC. Guido remains Python’s principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see <https://www.cnri.reston.va.us/>) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see <http://www.zope.com/>). In 2001, the Python Software Foundation (PSF, see <https://www.python.org/psf/>) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see <https://opensource.org/> for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

# Dealing with Bugs

Python is a mature programming language which has established a reputation for stability. In order to maintain this reputation, the developers would like to know of any deficiencies you find in Python.

It can be sometimes faster to fix bugs yourself and contribute patches to Python as it streamlines the process and involves less people. Learn how to [contribute](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/bugs.html#contributing-to-python).

## Using the Python issue tracker

Bug reports for Python itself should be submitted via the Python Bug Tracker (<https://bugs.python.org/>). The bug tracker offers a Web form which allows pertinent information to be entered and submitted to the developers.

The first step in filing a report is to determine whether the problem has already been reported. The advantage in doing so, aside from saving the developers time, is that you learn what has been done to fix it; it may be that the problem has already been fixed for the next release, or additional information is needed (in which case you are welcome to provide it if you can!). To do this, search the bug database using the search box on the top of the page.

If the problem you’re reporting is not already in the bug tracker, go back to the Python Bug Tracker and log in. If you don’t already have a tracker account, select the “Register” link or, if you use OpenID, one of the OpenID provider logos in the sidebar. It is not possible to submit a bug report anonymously.

Being now logged in, you can submit a bug. Select the “Create New” link in the sidebar to open the bug reporting form.

The submission form has a number of fields. For the “Title” field, enter a very short description of the problem; less than ten words is good. In the “Type” field, select the type of your problem; also select the “Component” and “Versions” to which the bug relates.

In the “Comment” field, describe the problem in detail, including what you expected to happen and what did happen. Be sure to include whether any extension modules were involved, and what hardware and software platform you were using (including version information as appropriate).

Each bug report will be assigned to a developer who will determine what needs to be done to correct the problem. You will receive an update each time action is taken on the bug.

## Tkinter Modules

Most of the time, [tkinter](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tkinter.html#module-tkinter) is all you really need, but a number of additional modules are available as well. The Tk interface is located in a binary module named \_tkinter. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

In addition to the Tk interface module, [tkinter](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tkinter.html#module-tkinter) includes a number of Python modules, tkinter.constants being one of the most important. Importing [tkinter](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tkinter.html#module-tkinter) will automatically import tkinter.constants, so, usually, to use Tkinter all you need is a simple import statement:

import tkinter

Or, more often:

from tkinter import \*

class tkinter.Tk(screenName=None, baseName=None, className='Tk', useTk=1)

The [Tk](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tkinter.html#tkinter.Tk) class is instantiated without arguments. This creates a toplevel widget of Tk which usually is the main window of an application. Each instance has its own associated Tcl interpreter.

tkinter.Tcl(screenName=None, baseName=None, className='Tk', useTk=0)

The [Tcl()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tkinter.html#tkinter.Tcl) function is a factory function which creates an object much like that created by the [Tk](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tkinter.html#tkinter.Tk) class, except that it does not initialize the Tk subsystem. This is most often useful when driving the Tcl interpreter in an environment where one doesn’t want to create extraneous toplevel windows, or where one cannot (such as Unix/Linux systems without an X server). An object created by the [Tcl()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tkinter.html#tkinter.Tcl) object can have a Toplevel window created (and the Tk subsystem initialized) by calling its loadtk() method.

Other modules that provide Tk support include:

[tkinter.scrolledtext](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tkinter.scrolledtext.html#module-tkinter.scrolledtext)

Text widget with a vertical scroll bar built in.

tkinter.colorchooser

Dialog to let the user choose a color.

tkinter.commondialog

Base class for the dialogs defined in the other modules listed here.

tkinter.filedialog

Common dialogs to allow the user to specify a file to open or save.

tkinter.font

Utilities to help work with fonts.

tkinter.messagebox

Access to standard Tk dialog boxes.

tkinter.simpledialog

Basic dialogs and convenience functions.

tkinter.dnd

Drag-and-drop support for [tkinter](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tkinter.html#module-tkinter). This is experimental and should become deprecated when it is replaced with the Tk DND.

[turtle](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/turtle.html#module-turtle)

Turtle graphics in a Tk window.

[**sqlite3**](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sqlite3.html#module-sqlite3) **— DB-API 2.0 interface for SQLite databases**

**Source code:** [Lib/sqlite3/](https://github.com/python/cpython/tree/3.8/Lib/sqlite3/)

SQLite is a C library that provides a lightweight disk-based database that doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It’s also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

The sqlite3 module was written by Gerhard Häring. It provides a SQL interface compliant with the DB-API 2.0 specification described by [**PEP 249**](https://www.python.org/dev/peps/pep-0249).

To use the module, you must first create a [Connection](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sqlite3.html#sqlite3.Connection) object that represents the database. Here the data will be stored in the example.db file:

import sqlite3

conn = sqlite3.connect('example.db')

You can also supply the special name :memory: to create a database in RAM.

Once you have a [Connection](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sqlite3.html#sqlite3.Connection), you can create a [Cursor](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sqlite3.html#sqlite3.Cursor) object and call its [execute()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sqlite3.html#sqlite3.Cursor.execute) method to perform SQL commands:

c = conn.cursor()

# Create table

c.execute('''CREATE TABLE stocks

(date text, trans text, symbol text, qty real, price real)''')

# Insert a row of data

c.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','RHAT',100,35.14)")

# Save (commit) the changes

conn.commit()

# We can also close the connection if we are done with it.

# Just be sure any changes have been committed or they will be lost.

conn.close()

The data you’ve saved is persistent and is available in subsequent sessions:

import sqlite3

conn = sqlite3.connect('example.db')

c = conn.cursor()

Usually your SQL operations will need to use values from Python variables. You shouldn’t assemble your query using Python’s string operations because doing so is insecure; it makes your program vulnerable to an SQL injection attack (see <https://xkcd.com/327/> for humorous example of what can go wrong).

Instead, use the DB-API’s parameter substitution. Put ? as a placeholder wherever you want to use a value, and then provide a tuple of values as the second argument to the cursor’s [execute()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sqlite3.html#sqlite3.Cursor.execute) method. (Other database modules may use a different placeholder, such as %s or :1.) For example:

# Never do this -- insecure!

symbol = 'RHAT'

c.execute("SELECT \* FROM stocks WHERE symbol = '%s'" % symbol)

# Do this instead

t = ('RHAT',)

c.execute('SELECT \* FROM stocks WHERE symbol=?', t)

print(c.fetchone())

# Larger example that inserts many records at a time

purchases = [('2006-03-28', 'BUY', 'IBM', 1000, 45.00),

('2006-04-05', 'BUY', 'MSFT', 1000, 72.00),

('2006-04-06', 'SELL', 'IBM', 500, 53.00),

]

c.executemany('INSERT INTO stocks VALUES (?,?,?,?,?)', purchases)

To retrieve data after executing a SELECT statement, you can either treat the cursor as an [iterator](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-iterator), call the cursor’s [fetchone()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sqlite3.html#sqlite3.Cursor.fetchone) method to retrieve a single matching row, or call [fetchall()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sqlite3.html#sqlite3.Cursor.fetchall) to get a list of the matching rows.

This example uses the iterator form:

>>> for row in c.execute('SELECT \* FROM stocks ORDER BY price'):

print(row)

('2006-01-05', 'BUY', 'RHAT', 100, 35.14)

('2006-03-28', 'BUY', 'IBM', 1000, 45.0)

('2006-04-06', 'SELL', 'IBM', 500, 53.0)

('2006-04-05', 'BUY', 'MSFT', 1000, 72.0)

# [os](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#module-os) — Miscellaneous operating system interfaces

**Source code:** [Lib/os.py](https://github.com/python/cpython/tree/3.8/Lib/os.py)

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see [open()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/functions.html#open), if you want to manipulate paths, see the [os.path](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.path.html#module-os.path) module, and if you want to read all the lines in all the files on the command line see the [fileinput](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/fileinput.html#module-fileinput) module. For creating temporary files and directories see the [tempfile](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/tempfile.html#module-tempfile) module, and for high-level file and directory handling see the [shutil](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/shutil.html#module-shutil) module.

Notes on the availability of these functions:

* The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface; for example, the function os.stat(path) returns stat information about path in the same format (which happens to have originated with the POSIX interface).
* Extensions peculiar to a particular operating system are also available through the [os](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#module-os) module, but using them is of course a threat to portability.
* All functions accepting path or file names accept both bytes and string objects, and result in an object of the same type, if a path or file name is returned.
* On VxWorks, os.fork, os.execv and os.spawn\*p\* are not supported.

Note

All functions in this module raise [OSError](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/exceptions.html#OSError) (or subclasses thereof) in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system.

exception os.error

An alias for the built-in [OSError](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/exceptions.html#OSError) exception.

os.name

The name of the operating system dependent module imported. The following names have currently been registered: 'posix', 'nt', 'java'.

See also

[sys.platform](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sys.html#sys.platform) has a finer granularity. [os.uname()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.uname) gives system-dependent version information.

The [platform](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/platform.html#module-platform) module provides detailed checks for the system’s identity.

## File Names, Command Line Arguments, and Environment Variables

In Python, file names, command line arguments, and environment variables are represented using the string type. On some systems, decoding these strings to and from bytes is necessary before passing them to the operating system. Python uses the file system encoding to perform this conversion (see [sys.getfilesystemencoding()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sys.html#sys.getfilesystemencoding)).

Changed in version 3.1: On some systems, conversion using the file system encoding may fail. In this case, Python uses the [surrogateescape encoding error handler](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/codecs.html#surrogateescape), which means that undecodable bytes are replaced by a Unicode character U+DCxx on decoding, and these are again translated to the original byte on encoding.

The file system encoding must guarantee to successfully decode all bytes below 128. If the file system encoding fails to provide this guarantee, API functions may raise UnicodeErrors.

## Process Parameters

These functions and data items provide information and operate on the current process and user.

os.ctermid()

Return the filename corresponding to the controlling terminal of the process.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix.

os.environ

A [mapping](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-mapping) object representing the string environment. For example, environ['HOME'] is the pathname of your home directory (on some platforms), and is equivalent to getenv("HOME") in C.

This mapping is captured the first time the [os](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#module-os) module is imported, typically during Python startup as part of processing site.py. Changes to the environment made after this time are not reflected in os.environ, except for changes made by modifying os.environ directly.

If the platform supports the [putenv()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.putenv) function, this mapping may be used to modify the environment as well as query the environment. [putenv()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.putenv) will be called automatically when the mapping is modified.

On Unix, keys and values use [sys.getfilesystemencoding()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sys.html#sys.getfilesystemencoding) and 'surrogateescape' error handler. Use [environb](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.environb) if you would like to use a different encoding.

Note

Calling [putenv()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.putenv) directly does not change os.environ, so it’s better to modify os.environ.

Note

On some platforms, including FreeBSD and Mac OS X, setting environ may cause memory leaks. Refer to the system documentation for putenv().

If [putenv()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.putenv) is not provided, a modified copy of this mapping may be passed to the appropriate process-creation functions to cause child processes to use a modified environment.

If the platform supports the [unsetenv()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.unsetenv) function, you can delete items in this mapping to unset environment variables. [unsetenv()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.unsetenv) will be called automatically when an item is deleted from os.environ, and when one of the pop() or clear() methods is called.

os.environb

Bytes version of [environ](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.environ): a [mapping](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-mapping) object representing the environment as byte strings. [environ](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.environ) and [environb](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.environb) are synchronized (modify [environb](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.environb) updates [environ](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.environ), and vice versa).

[environb](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.environb) is only available if [supports\_bytes\_environ](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.supports_bytes_environ) is True.

New in version 3.2.

os.chdir(path)

os.fchdir(fd)

os.getcwd()

These functions are described in [Files and Directories](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os-file-dir).

os.fsencode(filename)

Encode [path-like](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-path-like-object) filename to the filesystem encoding with 'surrogateescape' error handler, or 'strict' on Windows; return [bytes](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/stdtypes.html#bytes) unchanged.

[fsdecode()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.fsdecode) is the reverse function.

New in version 3.2.

Changed in version 3.6: Support added to accept objects implementing the [os.PathLike](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.PathLike) interface.

os.fsdecode(filename)

Decode the [path-like](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-path-like-object) filename from the filesystem encoding with 'surrogateescape' error handler, or 'strict' on Windows; return [str](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/stdtypes.html#str) unchanged.

[fsencode()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.fsencode) is the reverse function.

New in version 3.2.

Changed in version 3.6: Support added to accept objects implementing the [os.PathLike](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.PathLike) interface.

os.fspath(path)

Return the file system representation of the path.

If [str](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/stdtypes.html#str) or [bytes](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/stdtypes.html#bytes) is passed in, it is returned unchanged. Otherwise [\_\_fspath\_\_()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.PathLike.__fspath__) is called and its value is returned as long as it is a [str](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/stdtypes.html#str) or [bytes](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/stdtypes.html#bytes) object. In all other cases, [TypeError](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/exceptions.html#TypeError) is raised.

New in version 3.6.

class os.PathLike

An [abstract base class](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-abstract-base-class) for objects representing a file system path, e.g. [pathlib.PurePath](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/pathlib.html#pathlib.PurePath).

New in version 3.6.

abstractmethod \_\_fspath\_\_()

Return the file system path representation of the object.

The method should only return a [str](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/stdtypes.html#str) or [bytes](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/stdtypes.html#bytes) object, with the preference being for [str](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/stdtypes.html#str).

os.getenv(key, default=None)

Return the value of the environment variable key if it exists, or default if it doesn’t. key, default and the result are str.

On Unix, keys and values are decoded with [sys.getfilesystemencoding()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sys.html#sys.getfilesystemencoding) and 'surrogateescape' error handler. Use [os.getenvb()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.getenvb) if you would like to use a different encoding.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): most flavors of Unix, Windows.

os.getenvb(key, default=None)

Return the value of the environment variable key if it exists, or default if it doesn’t. key, default and the result are bytes.

[getenvb()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.getenvb) is only available if [supports\_bytes\_environ](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.supports_bytes_environ) is True.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): most flavors of Unix.

New in version 3.2.

os.get\_exec\_path(env=None)

Returns the list of directories that will be searched for a named executable, similar to a shell, when launching a process. env, when specified, should be an environment variable dictionary to lookup the PATH in. By default, when env is None, [environ](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.environ) is used.

New in version 3.2.

os.getegid()

Return the effective group id of the current process. This corresponds to the “set id” bit on the file being executed in the current process.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix.

os.geteuid()

Return the current process’s effective user id.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix.

os.getgid()

Return the real group id of the current process.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix.

os.getgrouplist(user, group)

Return list of group ids that user belongs to. If group is not in the list, it is included; typically, group is specified as the group ID field from the password record for user.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix.

New in version 3.3.

os.getgroups()

Return list of supplemental group ids associated with the current process.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix.

Note

On Mac OS X, [getgroups()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.getgroups) behavior differs somewhat from other Unix platforms. If the Python interpreter was built with a deployment target of 10.5 or earlier, [getgroups()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.getgroups) returns the list of effective group ids associated with the current user process; this list is limited to a system-defined number of entries, typically 16, and may be modified by calls to [setgroups()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.setgroups) if suitably privileged. If built with a deployment target greater than 10.5, [getgroups()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.getgroups) returns the current group access list for the user associated with the effective user id of the process; the group access list may change over the lifetime of the process, it is not affected by calls to [setgroups()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.setgroups), and its length is not limited to 16. The deployment target value, MACOSX\_DEPLOYMENT\_TARGET, can be obtained with [sysconfig.get\_config\_var()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/sysconfig.html#sysconfig.get_config_var).

os.getlogin()

Return the name of the user logged in on the controlling terminal of the process. For most purposes, it is more useful to use [getpass.getuser()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/getpass.html#getpass.getuser) since the latter checks the environment variables LOGNAME or USERNAME to find out who the user is, and falls back to pwd.getpwuid(os.getuid())[0] to get the login name of the current real user id.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix, Windows.

os.getpgid(pid)

Return the process group id of the process with process id pid. If pid is 0, the process group id of the current process is returned.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix.

os.getpgrp()

Return the id of the current process group.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix.

os.getpid()

Return the current process id.

os.getppid()

Return the parent’s process id. When the parent process has exited, on Unix the id returned is the one of the init process (1), on Windows it is still the same id, which may be already reused by another process.

[Availability](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/intro.html#availability): Unix, Windows.

Changed in version 3.2: Added support for Windows.

os.getpriority(which, who)

Get program scheduling priority. The value which is one of [PRIO\_PROCESS](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.PRIO_PROCESS), [PRIO\_PGRP](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.PRIO_PGRP), or [PRIO\_USER](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.PRIO_USER), and who is interpreted relative to which (a process identifier for [PRIO\_PROCESS](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.PRIO_PROCESS), process group identifier for [PRIO\_PGRP](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.PRIO_PGRP), and a user ID for [PRIO\_USER](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/os.html#os.PRIO_USER)). A zero value for who denotes (respectively) the calling process, the process group of the calling process, or the real user ID of the calling process.

class

A template for creating user-defined objects. Class definitions normally contain method definitions which operate on instances of the class.

class variable

A variable defined in a class and intended to be modified only at class level (i.e., not in an instance of the class).

coercion

The implicit conversion of an instance of one type to another during an operation which involves two arguments of the same type. For example, int(3.15) converts the floating point number to the integer 3, but in 3+4.5, each argument is of a different type (one int, one float), and both must be converted to the same type before they can be added or it will raise a [TypeError](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/exceptions.html#TypeError). Without coercion, all arguments of even compatible types would have to be normalized to the same value by the programmer, e.g., float(3)+4.5 rather than just 3+4.5.

complex number

An extension of the familiar real number system in which all numbers are expressed as a sum of a real part and an imaginary part. Imaginary numbers are real multiples of the imaginary unit (the square root of -1), often written i in mathematics or j in engineering. Python has built-in support for complex numbers, which are written with this latter notation; the imaginary part is written with a j suffix, e.g., 3+1j. To get access to complex equivalents of the [math](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/math.html#module-math) module, use [cmath](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/cmath.html#module-cmath). Use of complex numbers is a fairly advanced mathematical feature. If you’re not aware of a need for them, it’s almost certain you can safely ignore them.

context manager

An object which controls the environment seen in a [with](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/compound_stmts.html#with) statement by defining [\_\_enter\_\_()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/datamodel.html#object.__enter__) and [\_\_exit\_\_()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/datamodel.html#object.__exit__) methods. See [**PEP 343**](https://www.python.org/dev/peps/pep-0343).

context variable

A variable which can have different values depending on its context. This is similar to Thread-Local Storage in which each execution thread may have a different value for a variable. However, with context variables, there may be several contexts in one execution thread and the main usage for context variables is to keep track of variables in concurrent asynchronous tasks. See [contextvars](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/contextvars.html#module-contextvars).

contiguous

A buffer is considered contiguous exactly if it is either *C-contiguous* or *Fortran contiguous*. Zero-dimensional buffers are C and Fortran contiguous. In one-dimensional arrays, the items must be laid out in memory next to each other, in order of increasing indexes starting from zero. In multidimensional C-contiguous arrays, the last index varies the fastest when visiting items in order of memory address. However, in Fortran contiguous arrays, the first index varies the fastest.

coroutine

Coroutines are a more generalized form of subroutines. Subroutines are entered at one point and exited at another point. Coroutines can be entered, exited, and resumed at many different points. They can be implemented with the [async def](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/compound_stmts.html#async-def) statement. See also [**PEP 492**](https://www.python.org/dev/peps/pep-0492).

coroutine function

A function which returns a [coroutine](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-coroutine) object. A coroutine function may be defined with the [async def](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/compound_stmts.html#async-def) statement, and may contain [await](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/expressions.html#await), [async for](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/compound_stmts.html#async-for), and [async with](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/compound_stmts.html#async-with) keywords. These were introduced by [**PEP 492**](https://www.python.org/dev/peps/pep-0492).

CPython

The canonical implementation of the Python programming language, as distributed on [python.org](https://www.python.org). The term “CPython” is used when necessary to distinguish this implementation from others such as Jython or IronPython.

decorator

A function returning another function, usually applied as a function transformation using the @wrapper syntax. Common examples for decorators are [classmethod()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/functions.html#classmethod) and [staticmethod()](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/functions.html#staticmethod).

The decorator syntax is merely syntactic sugar, the following two function definitions are semantically equivalent:

def f(...):

...

f = staticmethod(f)

@staticmethod

def f(...):

...

The same concept exists for classes, but is less commonly used there. See the documentation for [function definitions](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/compound_stmts.html#function) and [class definitions](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/compound_stmts.html#class) for more about decorators.

function

A series of statements which returns some value to a caller. It can also be passed zero or more [arguments](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-argument) which may be used in the execution of the body. See also [parameter](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-parameter), [method](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-method), and the [Function definitions](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/compound_stmts.html#function) section.

function annotation

An [annotation](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-annotation) of a function parameter or return value.

Function annotations are usually used for [type hints](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-type-hint): for example, this function is expected to take two [int](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/functions.html#int) arguments and is also expected to have an [int](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/functions.html#int) return value:

def sum\_two\_numbers(a: int, b: int) -> int:

return a + b

Function annotation syntax is explained in section [Function definitions](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/reference/compound_stmts.html#function).

See [variable annotation](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/glossary.html#term-variable-annotation) and [**PEP 484**](https://www.python.org/dev/peps/pep-0484), which describe this functionality.

\_\_future\_\_

A pseudo-module which programmers can use to enable new language features which are not compatible with the current interpreter.

By importing the [\_\_future\_\_](mk:@MSITStore:C:\Users\JOKER%20PRINCE\AppData\Local\Programs\Python\Python38-32\Doc\Python380.chm::/library/__future__.html#module-__future__) module and evaluating its variables, you can see when a new feature was first added to the language and when it becomes the default:

>>> import \_\_future\_\_

>>> \_\_future\_\_.division

\_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192)