Python Standard Library

Builtins

Provides direct access to all built-in identifiers of Python.

```
import builtins


def str(value):
    return f"String {value}"


print(str(1), builtins.str(1))
```

```
import builtins


print("open" in dir(builtins))
print("copy" in dir(builtins))
print("exit" in dir(builtins))
```

Data types

`collections` - specialized container datatypes providing alternatives to Python's general purpose built-in containers.

```
# from collections import defaultdict
# from collections import OrderedDict
# from collections import namedtuple

# Queues

from collections import deque  # double-ended queue
# Alternative to queue.Queue and queue.LifoQueue (thread safe)
# and asyncio.Queue, asyncio.LifQueue, asyncio.PriorityQueue


queue = deque()

for i in range(5): queue.append(i)

print(queue.pop())
print(queue.popleft())
print(queue)
```

```
4
0
deque([1, 2, 3])
```

```
from array import array


example_array = array('l', [1, 2, 3, 4, 5])  # l - signed long
print(example_array)
print(example_array[0])
```

```
array('l', [1, 2, 3, 4, 5])
1
```

```
from heapq import heappop, heappush  # priority queue algorithm


h = []
heappush(h, (10, "cat"))
heappush(h, (1, "lion"))
heappush(h, (2, "dog"))

print(heappop(h))
```

```
(1, 'lion')
```

```
from enum import Enum, auto  # enumeration types


class ExampleEnum(Enum):
    FIRST = auto()
    SECOND = auto()


print(ExampleEnum.FIRST)
print(ExampleEnum(1))
```

```
ExampleEnum.FIRST
ExampleEnum.FIRST
```

Functional programming

```
from functools import singledispatch


@singledispatch
def myfunc(arg):
    print(f"Default {arg}")


@myfunc.register(int)
def myfunc_int(arg):
    print(f"Int {arg}")


@myfunc.register(list)
def myfunc_list(arg):
    print(f"List {arg}")


myfunc('s')
myfunc(1)
myfunc([1])
```

```
Default s
Int 1
List [1]
```

```
from itertools import chain


for i in chain([1, 2], [3, 4]):
    print(i)
```

```
1
2
3
4
```

```
from collections import ChainMap


map1 = {"a": 1, "b": 2}
map2 = {"c": 3, "d": 4}

for k, v in ChainMap(map1, map2).items():
    print(k, v)
```

```
from itertools import islice


# list(range(100))[:5]
islice(range(100), 5)  # first 5
islice(range(100), 5, 10)  # 5 to 10
for i in islice(range(100), 0, 40, 10):  # 0, 10, 20, ...
    print(i)
```

```
0
10
20
30
```

```
from itertools import dropwhile  # takewhile
```

```
for i in dropwhile(lambda x: x < 10, [-1, 1, 10, 100]):
  print(i)
```

```
10
100
```

```
from itertools import cycle, compress


pattern = cycle([False, False, True])
data = range(10)

for i in compress(data, pattern):
  print(i)
```

```
2
5
8
```

```
from itertools import accumulate


scores = [85, 92, 78, 96, 88]
for i in accumulate(scores, max):
  print(i)
```

```
85
92
92
96
96
```

```
from itertools import combinations, permutations, product


print(list(product(["a", "b"], ["0", "1"])))

print(list(permutations([1, 2, 3], 2)))

print(list(combinations([1, 2, 3], 2)))
```

```
[('a', '0'), ('a', '1'), ('b', '0'), ('b', '1')]
[(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)]
[(1, 2), (1, 3), (2, 3)]
```

Files

```
import pathlib  # Python 3.4: Object-oriented filesystem paths.


f1 = pathlib.Path("/content/sample_data/example1.txt")
f2 = f1.parent / "example2.txt"

print(f1.read_text())

with f2.open("r") as f:
    print(f.readlines())
```

```
import codecs  # manages encoding/decoding
import io


#


buffer = io.StringIO()
stream = codecs.getwriter('rot-13')(buffer)

text = 'example text'

stream.write(text)
stream.flush()
print(buffer.getvalue())
```

```
rknzcyr grkg
```

```
import tempfile  # Creating temporary files with unique names securely
from io import BytesIO


# in-memory temp file, until the size reaches max size
with tempfile.SpooledTemporaryFile(max_size=100, mode='w+t', encoding="utf-8") as temp:
    temp.write("Hi!")
    temp.seek(0)
    print(temp.read())
    print(temp.name)


f = BytesIO(b"Example!")
print(f.read())
```

```
Hi!
None
b'Example!'
```

```
import glob  # Filename pattern matching.


for name in sorted(glob.glob('/content/sample_data/*')):
    print(name)


import shutil # High-level file operations, such as file and directory copy, creating and extracting archives.


import linecache  # Randomly access files by line number. traceback uses it.


linecache.getline(linecache.__file__, 8)


import filecmp


dc = filecmp.cmp('/content/sample_data/example1.txt', '/content/sample_data/example2.txt')
print(dc)

# dc = filecmp.dircmp('example/dir1', 'example/dir2')
# dc.report()


# import tarfile, zipfile, hashlib
import zlib  # GNU zlib compression


data = b"1234568910"
compressed = zlib.compress(data, level=5)  # levels 0-9
checksum = zlib.adler32(data)  # adler32, crc32

print(compressed, checksum)
```

```
b'x^342615\xb3\xb044\x00\x00\x0b#\x02\x08' 186843656
```

```
# import selectors, socket, socketserver
```

Searialization / deserialization

```
# import csv, pickle, dbm, base64
import shelve  # A “shelf” is a persistent, dictionary-like object.


with shelve.open("/content/sample_data/test_shelf.db") as s:
    s["key1"] = {
        "int": 10,
        "s": "string",
    }


with shelve.open("/content/sample_data/test_shelf.db") as s:
    print(s["key1"])
```

```
{'int': 10, 's': 'string'}
```

```
# from xml.etree import ElementTree
from xml import sax  # Simple API for XML (SAX)
```

```
class BookHandler(sax.ContentHandler):
    def __init__(self):
        pass

    def startElement(self, name, attrs):
        pass

    def characters(self, content):
        pass

    def endElement(self, name):
        pass


# handler = BookHandler()
# sax.parseString("<xml></xml>", handler)
```

```
import json


data = {"a": 1, "b": 2}
print(json.dumps(data, indent=2))

print(json.dumps(data, separators=(',', ':')))

# echo '{"json":"obj"}' | python -m json.tool
# {
#     "json": "obj"
# }
```

```
{
  "a": 1,
  "b": 2
}
{"a":1,"b":2}
```

```
import tomllib  # Python 3.11


toml_str = """
python-version = "3.11.0"
python-implementation = "CPython"
"""

data = tomllib.loads(toml_str)
print(data)
```

```
{'python-version': '3.11.0', 'python-implementation': 'CPython'}
```

Concurrency

```
# import subprocess, threading, asyncio
import concurrent
import time


def task(t):
    time.sleep(t)
    print("Done")
    return t * 10


with concurrent.futures.ThreadPoolExecutor(max_workers=2) as ex:
    results = ex.map(task, [1, 2, 3])

    for res in results:
      print(res)
```

```
Done
10
Done
20
Done
30
```

Math

```
import math  # math.floor, math.ceil


print(math.radians(90))
```

```
1.5707963267948966
```

```
import statistics


print("mean =", statistics.mean([10, 5, 4, 2, 7, 9]))
print("mode =", statistics.mode([10, 5, 4, 2, 7, 9, 2]))
print("median =", statistics.median([10, 5, 4, 2, 7, 9]))
print("variance =", statistics.variance([10, 5, 4, 2, 7, 9]))
print("stdev =", statistics.stdev([10, 5, 4, 2, 7, 9]))
```

```
mean = 6.166666666666667
mode = 2
median = 6.0
variance = 9.366666666666667
stdev = 3.0605010483034745
```

```
# import decimal
import fractions


n1 = fractions.Fraction(16, 10)
print(n1)

n2 = fractions.Fraction(123)
print(n2)

n3 = fractions.Fraction("-0.25")
print(n3)

print(fractions.Fraction(1) / n3)
```

```
8/5
123
-1/4
-4
```

Contextlib

`ExitStack` - A context manager that is designed to make it easy to programmatically combine other context managers.

```
from contextlib import ExitStack, contextmanager


@contextmanager
def manager1():
    print("Entered 1")
    yield
    print("Exited 1")


@contextmanager
def manager2():
      print("Entered 2")
      yield
      print("Exited 2")


with manager1():
    with manager2():
        print("Example 1")


with (manager1(), manager2()):  # Python 3.10: Parenthesized context managers
    print("Example 2")


with ExitStack() as stack:
    stack.enter_context(manager1())
    stack.enter_context(manager2())

    print("Example ExitStack")
```

Regex

```
import re

NUMBER_RE = re.compile(r'(\d+)')


# alternative to findall
for i in NUMBER_RE.finditer('1, 100, text'):
  print(i)
```

Time

```
import time


print(time.time())
print(time.monotonic())  # return the value (in fractional seconds) of a monotonic clock, i.e. a clock that cannot go backwa


start = time.monotonic()
time.sleep(0.1)
dt = time.monotonic() - start

print(dt)


import datetime
import zoneinfo  # Python 3.9 IANA time zone support


dt = datetime.datetime(2020, 10, 31, 12, tzinfo=zoneinfo.ZoneInfo("Asia/Bangkok"))
print(dt)
print(dt.tzname())
```

sqlite3

```
import codecs
import sqlite3  # Embedded relational database. Implements a Python DB-API 2.0.


def decrypt(s):
    return codecs.encode(s, "rot-13")


with sqlite3.connect(":memory:") as conn:
    conn.create_function('decrypt', 1, decrypt)

    cursor = conn.cursor()

    cursor.execute("CREATE TABLE events(id int, name text);")
    cursor.execute("INSERT INTO events(id, name) VALUES (1, 'ThaiPy'), (2, 'PyCon');")

    cursor.execute("SELECT id, name, decrypt(name) FROM events WHERE name = :name", {"name": "ThaiPy"})

    for row in cursor.fetchall():
        print(row)
```

```
(1, 'ThaiPy', 'GunvCl')
```

Typing

```
from typing import Dict, TypedDict  # Python 3.8


d1: Dict[str, int] = {"a": 1, "b": 2}


class ExampleDict(TypedDict):
    name: str
    year: int
```

```
d2: ExampleDict = {"name": "Jane Doe", "year": 2000}
```

Venv

```
# python -m venv .venv
# source .venv/bin/activate
```

Skipped

smtplib, uuid, command line applications (argparse, password, readline, etc.), i18n and l10n, dev tools (timeit, trace), unittest, pdb, ABC, dataclasses, sys, os, platform, ...