

---

# Справочник по стандартной библиотеке

*Выпуск 3.8.8*

Гвидо ван Россум  
и команда разработчиков Python

августа 12, 2021

<https://Digitology.tech>  
Email: [tweakit@bk.ru](mailto:tweakit@bk.ru)



<b>1</b>	<b>Вступление</b>	<b>3</b>
1.1	Примечания по доступности . . . . .	4
<b>2</b>	<b>Встроенные функции</b>	<b>5</b>
<b>3</b>	<b>Встроенные константы</b>	<b>31</b>
3.1	Константы, добавленные модулем <code>site</code> . . . . .	32
<b>4</b>	<b>Встроенные типы</b>	<b>33</b>
4.1	Проверка истинности . . . . .	33
4.2	Логические операции — <code>and</code> , <code>or</code> , <code>not</code> . . . . .	34
4.3	Сравнения . . . . .	34
4.4	Числовые типы — <code>int</code> , <code>float</code> , <code>complex</code> . . . . .	35
4.5	Типы итераторов . . . . .	41
4.6	Типы последовательностей — <code>list</code> , <code>tuple</code> , <code>range</code> . . . . .	42
4.7	Тип текстовой последовательности — <code>str</code> . . . . .	49
4.8	Типы двоичных последовательностей — <code>bytes</code> , <code>bytearray</code> , <code>memoryview</code> . . . . .	61
4.9	Типы множества — <code>set</code> , <code>frozenset</code> . . . . .	86
4.10	Типы сопоставления — <code>dict</code> . . . . .	89
4.11	Типы менеджера контекста . . . . .	94
4.12	Другие встроенные типы . . . . .	95
4.13	Особые атрибуты . . . . .	97
<b>5</b>	<b>Встроенные исключения</b>	<b>99</b>
5.1	Базовые классы . . . . .	100
5.2	Конкретные исключения . . . . .	101
5.3	Предупреждения . . . . .	107
5.4	Иерархия исключений . . . . .	108
<b>6</b>	<b>Службы по обработке текста</b>	<b>109</b>
6.1	<code>string</code> — Общие строковые операции . . . . .	109
6.2	<code>re</code> — Операции с регулярными выражениями . . . . .	121
6.3	<code>difflib</code> — Хэлперы для вычисления различий . . . . .	144
6.4	<code>textwrap</code> — Обёртывание и заполнение текста . . . . .	156
6.5	<code>unicodedata</code> — База данных Юникод . . . . .	160
6.6	<code>stringprep</code> — Подготовка строк для Интернета . . . . .	162
6.7	<code>readline</code> — Интерфейс к GNU <code>readline</code> . . . . .	163

6.8	<code>rlcompleter</code> – Функция завершения для GNU readline . . . . .	168
<b>7</b>	<b>Бинарные данные</b>	<b>171</b>
7.1	<code>struct</code> – Интерпретация байтов как упакованные двоичные данные . . . . .	171
7.2	<code>codecs</code> – Реестр кодировок и базовых классов . . . . .	177
<b>8</b>	<b>Типы данных</b>	<b>197</b>
8.1	<code>datetime</code> – Базовые типы для представления даты и времени . . . . .	197
8.2	<code>calendar</code> – Календарные функции . . . . .	238
8.3	<code>collections</code> – Контейнерные типы данных . . . . .	242
8.4	<code>collections.abc</code> – Абстрактные базовые классы для контейнеров . . . . .	261
8.5	<code>heapq</code> – Алгоритм очереди кучи . . . . .	265
8.6	<code>bisect</code> – Алгоритм деления пополам . . . . .	270
8.7	<code>array</code> – Эффективные массивы числовых значений . . . . .	272
8.8	<code>weakref</code> – Слабые ссылки . . . . .	276
8.9	<code>types</code> – Динамическое создание типов и имена для встроенных типов . . . . .	283
8.10	<code>copy</code> – Функции поверхностного и глубокого копирования . . . . .	288
8.11	<code>pprint</code> – Приятная печать данных . . . . .	290
8.12	<code>reprlib</code> – Альтернативная реализация <code>repr()</code> . . . . .	296
8.13	<code>enum</code> – Поддержка перечислений . . . . .	298
<b>9</b>	<b>Числовые и математические модули</b>	<b>319</b>
9.1	<code>numbers</code> – Числовые абстрактные базовые классы . . . . .	319
9.2	<code>math</code> – Математические функции . . . . .	322
9.3	<code>cmath</code> – Математические функции для комплексных чисел . . . . .	329
9.4	<code>decimal</code> – Десятичная арифметика с фиксированной и плавающей точкой . . . . .	333
9.5	<code>fractions</code> – Рациональные числа . . . . .	362
9.6	<code>random</code> – Генерация псевдослучайных чисел . . . . .	365
9.7	<code>statistics</code> – Функции математической статистики . . . . .	372
<b>10</b>	<b>Модули функционального программирования</b>	<b>385</b>
10.1	<code>itertools</code> – Функции создания итераторов для эффективных циклов . . . . .	385
10.2	<code>functools</code> – Функции и операции высшего порядка над вызываемыми объектами . . . . .	401
10.3	<code>operator</code> – Стандартные операторы как функции . . . . .	411
<b>11</b>	<b>Доступ к файлам и каталогам</b>	<b>419</b>
11.1	<code>pathlib</code> – Объектно-ориентированные пути файловой системы . . . . .	419
11.2	<code>os.path</code> – Общие манипуляции с путями к файлам и каталогам . . . . .	437
11.3	<code>fileinput</code> – Перебор строк из нескольких входных потоков . . . . .	443
11.4	<code>stat</code> – Интерпретация результатов <code>stat()</code> . . . . .	445
11.5	<code>filecmp</code> – Сравнение файлов и каталогов . . . . .	451
11.6	<code>tempfile</code> – Генерация временных файлов и каталогов . . . . .	453
11.7	<code>glob</code> – Шаблоны расширений пути в стиле оболочки Unix . . . . .	458
11.8	<code>fnmatch</code> – Соответствие шаблону имени файла Unix . . . . .	459
11.9	<code>linecache</code> – Произвольный доступ к текстовым строкам . . . . .	460
11.10	<code>shutil</code> – Высокоуровневые файловые операции . . . . .	461
<b>12</b>	<b>Постоянство данных</b>	<b>473</b>
12.1	<code>pickle</code> – Сериализация Python объекта . . . . .	473
12.2	<code>copyreg</code> – Регистрация функций поддержки <code>pickle</code> . . . . .	491
12.3	<code>shelve</code> – Сохраняемость объектов Python . . . . .	492
12.4	<code>marshal</code> – Внутренняя сериализация объекта Python . . . . .	495
12.5	<code>dbm</code> – Интерфейсы для «баз данных» Unix . . . . .	496
12.6	<code>sqlite3</code> – Интерфейс DB-API 2.0 для баз данных SQLite . . . . .	501

<b>13</b>	<b>Компрессия данных и архивация</b>	<b>523</b>
13.1	zlib — Сжатие совместимое с <b>gzip</b>	523
13.2	gzip — Поддержка <b>gzip</b> файлов	527
13.3	bz2 — Поддержка сжатия <b>bzip2</b>	531
13.4	lzma — Сжатие с использованием алгоритма LZMA	535
13.5	zipfile — Работа с ZIP-архивами	541
13.6	tarfile — Чтение и запись файлов tar архива	551
<b>14</b>	<b>Форматы файлов</b>	<b>563</b>
14.1	csv — Чтение и запись CSV файлов	563
14.2	configparser — Парсер файла конфигурации	570
14.3	netrc — обработка файлов netrc	590
14.4	xdrlib — Кодирование и декодирование XDR данных	591
14.5	plistlib — Создание и парсинг файлов Mac OS .plist	594
<b>15</b>	<b>Криптографические сервисы</b>	<b>599</b>
15.1	hashlib — Безопасные хэши и дайджесты сообщений	599
15.2	hmac — Ключевое хеширование для аутентификации сообщений	610
15.3	secrets — Генерация безопасных случайных чисел для управления secrets	612
<b>16</b>	<b>Универсальные слжбы для различных операционных систем</b>	<b>615</b>
16.1	os — Разные интерфейсы к операционной системе	615
16.2	io — Основные инструменты для работы с потоками	672
16.3	time — Время доступа и конверсии	686
16.4	argparse — Парсер параметров командной строки, аргументов и подкоманд	697
16.5	getopt — С-подобный парсер параметров командной строки	730
16.6	logging — Логирование для Python	733
16.7	logging.config — Настройка логирования	751
16.8	logging.handlers — Обработчики логирования	762
16.9	getpass — Портатбельный ввод пароля	776
16.10	curses — Отрисовка псевдографического интерфейса в терминале	777
16.11	curses.textpad — Виджет текстового ввода для curses программ	796
16.12	curses.ascii — Утилиты для ASCII символов	798
16.13	curses.panel — Расширение стека panel для curses	800
16.14	platform — Доступ к данным идентификации базовой платформы	801
16.15	errno — Стандартные символы для errno	804
16.16	ctypes — Python библиотека внешних функций	810
<b>17</b>	<b>Конкурентное выполнение</b>	<b>847</b>
17.1	threading — Поточковый параллелизм	847
17.2	multiprocessing — Процессный параллелизм	861
17.3	multiprocessing.shared_memory — Предоставляет общую память для прямого доступа между процессами	908
17.4	Пакет concurrent	913
17.5	concurrent.futures — Запуск параллельных задач	913
17.6	subprocess — Управление подпроцессами	920
17.7	sched — Планировщик событий	940
17.8	queue — Синхронизированный класс очереди	941
17.9	_thread — Низкоуровневый API потоков	945
17.10	_dummy_thread — Замена модуля _thread	947
17.11	dummy_threading — Замена модуля threading	948
<b>18</b>	<b>contextvars — Контекстные переменные</b>	<b>949</b>
18.1	Переменные контекста	949
18.2	Ручное управление контекстом	950

18.3	Поддержка <code>asyncio</code> . . . . .	952
<b>19</b>	<b>Сетевое и межпроцессное взаимодействие</b>	<b>955</b>
19.1	<code>asyncio</code> – Асинхронный I/O . . . . .	955
19.2	<code>socket</code> – Низкоуровневый сетевой интерфейс . . . . .	1046
19.3	<code>ssl</code> – TLS/SSL обёртка для сокетных объектов . . . . .	1073
19.4	<code>select</code> – Ожидание завершения I/O . . . . .	1111
19.5	<code>selectors</code> – Высокоуровневое мультиплексирование ввода-вывода . . . . .	1119
19.6	<code>asyncore</code> – Асинхронный обработчик сокетов . . . . .	1122
19.7	<code>asynchat</code> – Асинхронный сокет обработчика запрос/ответ . . . . .	1127
19.8	<code>signal</code> – Установить обработчики для асинхронных событий . . . . .	1130
19.9	<code>mmap</code> – Поддержка отображаемых в память файлов . . . . .	1138
<b>20</b>	<b>Обработка интернет данных</b>	<b>1145</b>
20.1	<code>email</code> – Электронная почта и пакет обработки MIME . . . . .	1145
20.2	<code>json</code> – Кодер и декодер JSON . . . . .	1208
20.3	<code>mailcap</code> – Обработка файлов Mailcap . . . . .	1219
20.4	<code>mailbox</code> – Манипулирование почтовыми ящиками в различных форматах . . . . .	1220
20.5	<code>mimetypes</code> – Сопоставление имен файлов с MIME типами . . . . .	1239
20.6	<code>base64</code> – Base16, Base32, Base64, Base85 кодировки данных . . . . .	1242
20.7	<code>binhex</code> – Кодирование и декодирование binhex4 файлов . . . . .	1245
20.8	<code>binascii</code> – преобразования между двоичной и ASCII . . . . .	1246
20.9	<code>quopri</code> – Кодировать и декодировать MIME quoted-printable для печати . . . . .	1249
20.10	<code>uu</code> – Кодировать и декодировать файлы uuencode . . . . .	1249
<b>21</b>	<b>Инструменты обработки структурированной разметки</b>	<b>1251</b>
21.1	<code>html</code> – Поддержка языка гипертекстовой разметки . . . . .	1251
21.2	<code>html.parser</code> – Простой HTML и XHTML парсер . . . . .	1252
21.3	<code>html.entities</code> – Определения общих сущностей HTML . . . . .	1257
21.4	Модули обработки XML . . . . .	1257
21.5	<code>xml.etree.ElementTree</code> – ElementTree XML API . . . . .	1259
21.6	<code>xml.dom</code> – API объектной модели документа . . . . .	1279
21.7	<code>xml.dom.minidom</code> – Минимальная реализация DOM . . . . .	1290
21.8	<code>xml.dom.pulldom</code> – Поддержка построения частичных деревьев DOM . . . . .	1295
21.9	<code>xml.sax</code> – Поддержка SAX2 парсеров . . . . .	1297
21.10	<code>xml.sax.handler</code> – Базовые классы для обработчиков SAX . . . . .	1299
21.11	<code>xml.sax.saxutils</code> – Утилиты SAX . . . . .	1304
21.12	<code>xml.sax.xmlreader</code> – Интерфейс для XML парсеров . . . . .	1305
21.13	<code>xml.parsers.expat</code> – Быстрый парсинг XML с использованием Expat . . . . .	1309
<b>22</b>	<b>Интернет протоколы и поддержка</b>	<b>1321</b>
22.1	<code>webbrowser</code> – Удобный контроллер веб-браузера . . . . .	1321
22.2	<code>cgi</code> – Поддержка Общего Интерфейса Шлюза . . . . .	1324
22.3	<code>cgitb</code> – Трейсбэк менеджер для CGI скриптов . . . . .	1331
22.4	<code>wsgiref</code> – Утилиты WSGI и справочная реализация . . . . .	1332
22.5	<code>urllib</code> – Модули обработки URL-адресов . . . . .	1343
22.6	<code>urllib.request</code> – Расширяемая библиотека для открытия URL-адресов . . . . .	1343
22.7	<code>urllib.response</code> – Классы ответов, используемые urllib . . . . .	1363
22.8	<code>urllib.parse</code> – Разбор URL-адреса на компоненты . . . . .	1363
22.9	<code>urllib.error</code> – Классы исключений, созданные urllib.request . . . . .	1372
22.10	<code>urllib.robotparser</code> – Парсер для robots.txt . . . . .	1373
22.11	<code>http</code> – HTTP модули . . . . .	1374
22.12	<code>http.client</code> – Клиент протокола HTTP . . . . .	1376
22.13	<code>ftplib</code> – Клиент протокола FTP . . . . .	1383
22.14	<code>poplib</code> – Клиент протокола POP3 . . . . .	1389

22.15	imaplib	— Клиент протокола IMAP4	1392
22.16	nntplib	— Клиент протокола NNTP	1399
22.17	smtplib	— Клиент протокола SMTP	1407
22.18	smtpd	— SMTP сервер	1414
22.19	telnetlib	— Клиент Telnet	1417
22.20	uuid	— UUID объекты согласно RFC 4122	1421
22.21	socketserver	— Фреймворк для сетевых серверов	1424
22.22	http.server	— HTTP серверы	1433
22.23	http.cookies	— Управление состоянием HTTP	1440
22.24	http.cookiejar	— Обработка cookie для HTTP-клиентов	1443
22.25	xmlrpc	— Серверные и клиентские XMLRPC модули	1453
22.26	xmlrpc.client	— Клиентский доступ к XML-RPC	1453
22.27	xmlrpc.server	— Базовые серверы XML-RPC	1462
22.28	ipaddress	— Библиотека манипулирования IPv4/IPv6	1468
<b>23</b>	<b>Мультимедийные сервисы</b>		<b>1483</b>
23.1	audioop	— Манипуляция необработанными аудиоданными	1483
23.2	aifc	— чтение и запись AIFF и AIFC файлов	1487
23.3	sunau	— Чтение и запись файлов Sun AU	1489
23.4	wave	— Чтение и запись WAV файлов	1492
23.5	chunk	— Чтение IFF чанков данных	1494
23.6	colorsys	— Преобразования между цветовыми системами	1496
23.7	imghdr	— Определение типа изображения	1497
23.8	sndhdr	— Определите типа звукового файла	1497
23.9	ossaudiodev	— Доступ к OSS-совместимым аудиоустройствам	1498
<b>24</b>	<b>Интернационализация</b>		<b>1505</b>
24.1	gettext	— Многоязычные службы интернационализации	1505
24.2	locale	— Сервисы интернационализации	1515
<b>25</b>	<b>Программные фреймворки</b>		<b>1525</b>
25.1	turtle	— Черепашня графика	1525
25.2	cmd	— Поддержка линейно ориентированных командных интерпретаторов	1551
25.3	shlex	— Простой лексический анализ	1557
<b>26</b>	<b>Графические пользовательские интерфейсы с Tk</b>		<b>1563</b>
26.1	tkinter	— Python интерфейс для Tcl/Tk	1563
26.2	tkinter.ttk	— Стилизуемые виджеты Tk	1575
26.3	tkinter.tix	— Расширение виджетов для Tk	1594
26.4	tkinter.scrolledtext	— Прокручиваемый текстовый виджет	1599
26.5	IDLE		1599
26.6	Другие пакеты графического интерфейса пользователя		1611
<b>27</b>	<b>Средства разработки</b>		<b>1613</b>
27.1	typing	— Поддержка подсказок типа	1613
27.2	pydoc	— Генератор документации и интерактивная справочная система	1634
27.3	doctest	— Тестовые интерактивные примеры Python	1636
27.4	unittest	— Фреймворк юнит тестирования	1662
27.5	unittest.mock	— библиотека mock объектов	1695
27.6	unittest.mock	— приступающая к работе	1739
27.7	2to3	- автоматизированный перевод кода Python 2 на 3	1761
27.8	test	— Пакет регрессионных тестов для Python	1767
27.9	test.support	— Утилиты для набора тестов Python	1769
27.10	test.support.script_helper	— утилиты для выполнения тестов Python	1783

<b>28</b>	<b>Отладка и профилирование</b>	<b>1787</b>
28.1	Таблица аудита событий . . . . .	1787
28.2	<code>bdb</code> — Отладочный фреймворк . . . . .	1791
28.3	<code>faulthandler</code> — Дамп трассировки Python . . . . .	1796
28.4	<code>pdb</code> — Отладчик Python . . . . .	1798
28.5	Профилировщики Python . . . . .	1805
28.6	<code>timeit</code> — Измерение времени выполнения небольших фрагментов кода . . . . .	1815
28.7	<code>trace</code> — Трассировка или отслеживание выполнения инструкции Python . . . . .	1820
28.8	<code>tracemalloc</code> — Отслеживание выделения памяти . . . . .	1822
<b>29</b>	<b>Пакетизация и установка ПО</b>	<b>1833</b>
29.1	<code>distutils</code> — Сборка и установка Python модулей . . . . .	1833
29.2	<code>ensurepip</code> — Начальная загрузка ``pip`` установщика . . . . .	1834
29.3	<code>venv</code> — Создание виртуальных сред . . . . .	1835
29.4	<code>zipapp</code> — Управление исполняемыми zip-архивами Python . . . . .	1845
<b>30</b>	<b>Службы среды выполнения Python</b>	<b>1853</b>
30.1	<code>sys</code> — Параметры и функции, специфичные для системы . . . . .	1853
30.2	<code>sysconfig</code> — Предоставляет доступ к информации о конфигурации Python . . . . .	1874
30.3	<code>builtins</code> — Встроенные объекты . . . . .	1878
30.4	<code>__main__</code> — Сценарий верхнего уровня . . . . .	1879
30.5	<code>warnings</code> — Управление предупреждениями . . . . .	1879
30.6	<code>dataclasses</code> — Классы данных . . . . .	1886
30.7	<code>contextlib</code> — Утилиты для контекстов оператора <code>with</code> . . . . .	1895
30.8	Абстрактные базовые классы <code>abc</code> . . . . .	1909
30.9	<code>atexit</code> — Обработчики выхода . . . . .	1914
30.10	<code>traceback</code> — Распечатать или получить трассировку стека . . . . .	1916
30.11	<code>__future__</code> — Определения будущих инструкций . . . . .	1923
30.12	<code>gc</code> — Интерфейс к сборщику мусора . . . . .	1924
30.13	<code>inspect</code> — Осмотр живых объектов . . . . .	1928
30.14	<code>site</code> — Сайт-специфичный хук конфигурации . . . . .	1945
<b>31</b>	<b>Пользовательские интерпретаторы Python</b>	<b>1949</b>
31.1	<code>code</code> — Базовые классы интерпретатора . . . . .	1949
31.2	<code>codeop</code> — Компиляция Python кода . . . . .	1951
<b>32</b>	<b>Импорт модулей</b>	<b>1953</b>
32.1	<code>zipimport</code> — Импорт модулей из Zip архивов . . . . .	1953
32.2	<code>pkgutil</code> — Утилита расширения пакета . . . . .	1955
32.3	<code>modulefinder</code> — Найти модули, используемые скриптом . . . . .	1958
32.4	<code>runpy</code> — Поиск и выполнение Python модулей . . . . .	1960
32.5	<code>importlib</code> — Реализация <code>import</code> . . . . .	1962
32.6	Применение <code>importlib.metadata</code> . . . . .	1984
<b>33</b>	<b>Языковые сервисы Python</b>	<b>1989</b>
33.1	<code>parser</code> — Доступ к деревьям разбора Python . . . . .	1989
33.2	<code>ast</code> — Абстрактные синтаксические деревья . . . . .	1994
33.3	<code>symtable</code> — Доступ к таблицам символов компилятора . . . . .	2001
33.4	<code>symbol</code> — Константы, используемые с деревьями синтаксического анализа Python . . . . .	2003
33.5	<code>token</code> — Константы, используемые с деревьями разбора Python . . . . .	2004
33.6	<code>keyword</code> — Тестирование ключевых слов Python . . . . .	2007
33.7	<code>tokenize</code> — Токенизатор для исходного кода Python . . . . .	2008
33.8	<code>tabnanny</code> — Обнаружение неоднозначного отступа . . . . .	2012
33.9	<code>pyclbr</code> — Поддержка браузера Python модуля . . . . .	2013
33.10	<code>py_compile</code> — Компиляция исходных файлов Python . . . . .	2014



33.11	<code>compileall</code> – байт-компиляции библиотеки Python	2016
33.12	<code>dis</code> – Дизассемблер для Python байткода	2020
33.13	<code>pickletools</code> – Инструменты для pickle разработчиков	2035
<b>34</b>	<b>Разные сервисы</b>	<b>2037</b>
34.1	<code>formatter</code> – Общее форматирование вывода	2037
<b>35</b>	<b>Специальные службы MS Windows</b>	<b>2043</b>
35.1	<code>msilib</code> – Чтение и запись файлов установщика Microsoft	2043
35.2	<code>msvcrt</code> – Полезные функции из среды выполнения MS VC++	2049
35.3	<code>winreg</code> – Доступ к реестру Windows	2050
35.4	<code>winsound</code> – Интерфейс воспроизведения звука для Windows	2060
<b>36</b>	<b>Unix специфичные службы</b>	<b>2063</b>
36.1	<code>posix</code> – Наиболее распространенные системные вызовы POSIX	2063
36.2	<code>pwd</code> – База паролей	2064
36.3	<code>spwd</code> – База данных теневых паролей	2065
36.4	<code>grp</code> – База данных групп	2066
36.5	<code>crypt</code> – Функция проверки Unix паролей	2067
36.6	<code>termios</code> – POSIX стиль управления tty	2069
36.7	<code>tty</code> – Функции управления терминалом	2070
36.8	<code>pty</code> – Псевдо-терминальные утилиты	2070
36.9	<code>fcntl</code> – Системные вызовы <code>fcntl</code> и <code>ioctl</code>	2072
36.10	<code>pipes</code> – Интерфейс для pipelines оболочки	2074
36.11	<code>resource</code> – Информация об использовании ресурсов	2075
36.12	<code>nis</code> – Интерфейс для Sun NIS (Желтые страницы)	2080
36.13	<code>syslog</code> – Подпрограммы библиотеки <code>syslog</code> Unix	2081
<b>37</b>	<b>Замененные модули</b>	<b>2083</b>
37.1	<code>optparse</code> – Парсер для параметров командной строки	2083
37.2	<code>imp</code> – Доступ к внутренностям <code>import</code>	2112
<b>38</b>	<b>Недокументированные модули</b>	<b>2119</b>
38.1	Модули специфичные для платформы	2119
<b>A</b>	<b>Глоссарий</b>	<b>2121</b>
	<b>Литература</b>	<b>2137</b>
	<b>Содержание модулей Python</b>	<b>2139</b>
	<b>Алфавитный указатель</b>	<b>2143</b>



Хотя `reference-index` описывает точный синтаксис и семантику языка Python, в этом справочном руководстве описывается стандартная библиотека, которая распространяется вместе с Python. В нем также описаны некоторые необязательные компоненты, которые обычно включаются в Python дистрибутивы.

Стандартная библиотека Python очень обширна и предлагает широкий спектр возможностей, о чем свидетельствует подробное содержание, приведенное ниже. Библиотека содержит встроенные модули (написанные на языке C), обеспечивающие доступ к таким функциональным возможностям системы, как файловый ввод-вывод, который в противном случае был бы недоступен для Python программистов, а также модули, написанные на языке Python, обеспечивающие стандартные решения многих проблем, возникающих при повседневном программировании. Некоторые из этих модулей специально разработаны для поощрения и повышения переносимости Python программ путем абстракции особенностей платформы в виде нейтрального к платформе API.

Установщики Python для платформы Windows обычно включают всю стандартную библиотеку и часто также включают множество дополнительных компонентов. Для Unix-подобных операционных систем Python обычно предоставляется в виде набора пакетов, поэтому может потребоваться использование пакетных инструментов, поставляемые с операционной системой, для получения некоторых или всех необязательных компонентов.

Помимо стандартной библиотеки, существует растущая коллекция из нескольких тысяч компонентов (от отдельных программ и модулей до пакетов и фреймворков разработки приложений), доступных из [Пакетного Индекса Python](#).



---

## Вступление

---

«Библиотека Python» содержит несколько различных типов компонентов.

Она содержит типы данных, которые обычно считаются частью «ядра» языка, например числа и списки. Для этих типов ядро языка Python определяет форму литералов и накладывает некоторые ограничения на их семантику, но не полностью определяет семантику. (С другой стороны, ядро языка действительно определяет синтаксические свойства, такие как написание и приоритеты операторов.)

Библиотека также содержит встроенные функции и объекты исключений, которые могут использоваться любым Python кодом без использования оператора `import`. Некоторые из них определены базовым языком, но многие не являются существенными для базовой семантики и описаны только здесь.

Однако основная часть библиотеки состоит из набора модулей. Есть много способов проанализировать эту коллекцию. Некоторые модули написаны на C и встроены в интерпретатор Python; другие написаны на Python и импортированы в форме исходника. Некоторые модули предоставляют интерфейсы, которые очень специфичны для Python, например, печать трассировки стека; некоторые предоставляют интерфейсы, специфичные для конкретных операционных систем, такие как доступ к определенному оборудованию; другие предоставляют интерфейсы, специфичные для определенного домена приложения, например World Wide Web. Некоторые модули доступны во всех версиях и портах Python; другие доступны только в том случае, если базовая система поддерживает или требует их; все же другие доступны только в том случае, если во время компиляции и установки Python был выбран конкретный вариант конфигурации.

Это руководство организовано «изнутри»: сначала в нём описываются встроенные функции, типы данных и исключения, и, наконец, модули, сгруппированные по главам связанных модулей.

Это означает, что если вы начнете читать это руководство с самого начала и перейдете к следующей главе, когда вам станет скучно, вы получите разумный обзор доступных модулей и областей приложения, которые поддерживаются библиотекой Python. Конечно, вы не можете читать его как роман, вы также можете просматривать оглавление (перед руководством) или искать конкретную функцию, модуль или термин в указателе (сзади). И, наконец, если вам нравится изучать случайные предметы, вы выбираете случайный номер страницы (см. модуль *random*) и читаете один или два раздела. Независимо от того, в каком порядке вы читаете разделы данного руководства, лучше начать с главы *Встроенные функции*, т. к. оставшаяся часть руководства предполагает знакомство с этим материалом.

Пусть шоу начнется!

## 1.1 Примечания по доступности

- Примечание «Доступность: Unix» означает, что эта функция обычно встречается в системах Unix. Он не делает никаких заявлений о своём существовании в конкретной операционной системе.
- Если не указано отдельно, все функции, заявляющие «Доступность: Unix», поддерживаются в Mac OS X, построенной на ядре Unix.

## Встроенные функции

У Python интерпретатора есть ряд встроенных в него функций и типов, которые доступны всегда. Они перечислены далее в алфавитном порядке.

Встроенные функции				
<i>abs()</i>	<i>delattr()</i>	<i>hash()</i>	<i>memoryview()</i>	<i>set()</i>
<i>all()</i>	<i>dict()</i>	<i>help()</i>	<i>min()</i>	<i>setattr()</i>
<i>any()</i>	<i>dir()</i>	<i>hex()</i>	<i>next()</i>	<i>slice()</i>
<i>ascii()</i>	<i>divmod()</i>	<i>id()</i>	<i>object()</i>	<i>sorted()</i>
<i>bin()</i>	<i>enumerate()</i>	<i>input()</i>	<i>oct()</i>	<i>staticmethod()</i>
<i>bool()</i>	<i>eval()</i>	<i>int()</i>	<i>open()</i>	<i>str()</i>
<i>breakpoint()</i>	<i>exec()</i>	<i>isinstance()</i>	<i>ord()</i>	<i>sum()</i>
<i>bytearray()</i>	<i>filter()</i>	<i>issubclass()</i>	<i>pow()</i>	<i>super()</i>
<i>bytes()</i>	<i>float()</i>	<i>iter()</i>	<i>print()</i>	<i>tuple()</i>
<i>callable()</i>	<i>format()</i>	<i>len()</i>	<i>property()</i>	<i>type()</i>
<i>chr()</i>	<i>frozenset()</i>	<i>list()</i>	<i>range()</i>	<i>vars()</i>
<i>classmethod()</i>	<i>getattr()</i>	<i>locals()</i>	<i>repr()</i>	<i>zip()</i>
<i>compile()</i>	<i>globals()</i>	<i>map()</i>	<i>reversed()</i>	<i>__import__()</i>
<i>complex()</i>	<i>hasattr()</i>	<i>max()</i>	<i>round()</i>	

### **abs(x)**

Возвращает абсолютное значение числа. Аргумент может быть целым числом или числом с плавающей запятой. Если аргумент — комплексное число, возвращает его амплитуду. Если *x* определяет `__abs__()`, `abs(x)` возвращает `x.__abs__()`.

### **all(iterable)**

Возвращает `True`, если все элементы *iterable* истинны (или если итерация пуста). Эквивалентно:

```
def all(iterable):
    for element in iterable:
        if not element:
            return False
    return True
```

**any**(*iterable*)

Возвращает `True`, если какой-либо элемент *iterable* истинен. Если итерируемый объект пуст, возвращает `False`. Эквивалентно:

```
def any(iterable):
    for element in iterable:
        if element:
            return True
    return False
```

**ascii**(*object*)

Как и `repr()`, возвращает строку печатаемого представления объекта, но экранирует отличные от ASCII символы, в возвращаемой строке `repr()`, используя экранирование `\x`, `\u` или `\U`. Функция генерирует строку, аналогичную возвращаемой `repr()` в Python 2.

**bin**(*x*)

Преобразовать целое число в двоичную строку с префиксом «0b». Результатом является допустимое выражение Python. Если *x* не является объектом Python `int`, он должен определить метод `__index__()`, который возвращает целое число. Несколько примеров:

```
>>> bin(3)
'0b11'
>>> bin(-10)
'-0b1010'
```

Если нужен префикс «0b» или нет, вы можете использовать любой из следующих вариантов.

```
>>> format(14, '#b'), format(14, 'b')
('0b1110', '1110')
>>> f'{14:#b}', f'{14:b}'
('0b1110', '1110')
```

См. также `format()` для получения дополнительной информации.

**class bool**([*x*])

Возвращает логическое значение, например `True` или `False`. *x* конвертируется с использованием стандартной *процедуры проверки истинности*. Если *x* ложно или пропущено, возвращается `False`; в противном случае возвращается `True`. Класс `bool` является подклассом `int` (см. *Числовые типы — int, float, complex*). От него нельзя создавать подклассы. Его единственные экземпляры — `False` и `True` (см. *Логические значения*).

Изменено в версии 3.7: *x* теперь является только позиционным параметром.

**breakpoint**(*\*args, \*\*kws*)

Функция переводит вас в отладчик на месте вызова. В частности, она вызывает `sys.breakpointhook()`, прямо передав `args` и `kws`. По умолчанию `sys.breakpointhook()` вызывает `pdb.set_trace()`, не ожидая аргументов. В данном случае это чисто вспомогательная функция, поэтому вам не нужно явно импортировать `pdb` или вводить столько кода для входа в отладчик. Однако `sys.breakpointhook()` можно настроить на какую-либо другую функцию, и `breakpoint()` автоматически вызовет её, позволяя вам перейти в выбранный отладчик.

Raises an *auditing event* `builtins.breakpoint` with argument `breakpointhook`.

Добавлено в версии 3.7.

**class bytearray**([*source*[, *encoding*[, *errors*]]])

Возвращает новый массив байтов. Класс `bytearray` представляет собой изменяемую последовательность целых чисел в диапазоне  $0 \leq x < 256$ . У него большинство обычных методов изменяемых



последовательностей, описанных в *Типы изменяемых последовательностей*, а также большинство методов, у которых тип `bytes`, см. *Байты и операции с байтовыми массивами*.

Необязательный параметр `source` можно использовать для инициализации массива несколькими способами:

- Если это *string*, вы также должны указать параметры *encoding* (и, необязательно, *errors*); Затем `bytearray()` преобразует строку в байты, используя `str.encode()`.
- Если это *integer*, `У` массива будет такой размер и он будет инициализирован нулевыми байтами.
- Если это объект, соответствующий интерфейсу буфера, то для инициализации массива байтов будет использоваться буфер объекта, доступный только для чтения.
- Если это *iterable*, должен быть итерацией целых чисел в диапазоне  $0 \leq x < 256$ , которые используются в качестве начального содержимого массива.

Без аргумента создаётся массив размером 0.

См. также *Типы двоичных последовательностей* — `bytes`, `bytearray`, `memoryview` и *Объекты bytearray*.

**class bytes**(`[source[, encoding[, errors]]]`)

Возвращает новый объект «bytes», который представляет собой неизменяемую последовательность целых чисел в диапазоне  $0 \leq x < 256$ . `bytes` являются неизменной версией `bytearray` — у него те же немутантные методы и такое же поведение индексации и срезов.

Соответственно, аргументы конструктора интерпретируются как `bytearray()`.

Объекты байтов также можно создать с помощью литералов, см. `strings`.

См. также *Типы двоичных последовательностей* — `bytes`, `bytearray`, `memoryview`, *Объекты байтов* и *Байты и операции с байтовыми массивами*.

**callable(object)**

Возвращает `True`, если аргумент *object* поддерживает возможность вызова и `False`, если нет. Возвращённое `True` не гарантирует успешного вызова, но если `False`, вызов *object* никогда не будет успешным. Обратите внимание, что классы могут вызываться (вызов класса возвращает новый экземпляр); экземпляры вызываются, если у их класса есть метод `__call__()`.

Добавлено в версии 3.2: Функция была сначала удалена в Python 3.0, а затем возвращена в Python 3.2.

**chr(i)**

Возвращает строку (представляющую символ), кодовая точка которого в Юникоде равна целому числу *i*. Например, `chr(97)` возвращает строку `'a'`, а `chr(8364)` возвращает строку `'€'`. Обратная функции `ord()`.

Допустимый диапазон аргумента — от 0 до `1_114_111` (`0x10FFFF` по основанию 16). Будет поднято `ValueError`, если *i* находится вне этого диапазона.

**@classmethod**

Преобразует метод в метод класса.

Метод класса получает класс как неявный первый аргумент, точно так же, как метод экземпляра получает экземпляр. Чтобы объявить метод класса, используйте следующую идиому:

```
class C:
    @classmethod
    def f(cls, arg1, arg2, ...): ...
```

Форма `@classmethod` является функцией *декоратором*; подробности см. в `function`.

Метод класса может быть вызван либо для класса (например, `C.f()`), либо для экземпляра (например, `C().f()`). Экземпляр игнорируется, за исключением его класса. Если метод класса вызывается для производного класса, объект производного класса передается как подразумеваемый первый аргумент.

Методы класса отличаются от статических методов C++ или Java. Если они вам нужны, см. [`staticmethod\(\)`](#).

Дополнительные сведения о методах класса см. в [types](#).

**`compile(source, filename, mode, flags=0, dont_inherit=False, optimize=-1)`**

Скомпилировать `source` в код или AST объект. Кодовые объекты могут выполняться [`exec\(\)`](#) или [`eval\(\)`](#). `source` может быть обычной строкой, байтовой строкой или AST объектом. Обратитесь к документации модуля [`ast`](#) для получения информации о том, как работать с AST объектами.

Аргумент `filename` указывает файл, из которого был прочитан код; если код не был прочитан из файла, можно передать любое значение, например строку `'<string>'`.

Аргумент `mode` указывает режим компиляции кода; может быть `'exec'`, если `source` состоит из последовательности операторов, `'eval'`, если он состоит из одного выражения, или `'single'`, если он состоит из одного интерактивного оператора (в последнем случае будут напечатаны операторы выражений, которые что-то вычисляют, кроме `None`).

Необязательные аргументы `flags` и `dont_inherit` управляют тем, как будущие операторы влияют на компиляцию `source`. Если ни один из них не присутствует (или оба равны нулю), код компилируется с теми будущими операторами, которые действуют в коде, вызывающем [`compile\(\)`](#). Если задан аргумент `flags`, а `dont_inherit` не равен (или равен нулю), то будущие операторы, указанные аргументом `flags`, используются в дополнение к тем, которые будут использоваться в любом случае. Если `dont_inherit` — ненулевое целое число, тогда аргумент `flags` является им — будущие операторы, действующие вокруг вызова компиляции, игнорируются.

Будущие операторы задаются битами, которые можно объединить побитовым ИЛИ, чтобы указать несколько операторов. Битовое поле, необходимое для указания данной функции, можно найти как атрибут `compiler_flag` в экземпляре `_Feature` в модуле [`\_\_future\_\_`](#).

Необязательный аргумент `flags` также определяет, разрешено ли скомпилированному источнику содержать высокоуровневое `await`, `async for` и `async with`. Когда бит `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT` установлен, у объекта кода возврата будет `CO_COROUTINE`, установленный в `co_code`, и может интерактивно выполняться через `await eval(code_object)`.

Аргумент `optimize` указывает уровень оптимизации компилятора; значение по умолчанию `-1` выбирает уровень оптимизации интерпретатора в соответствии с параметрами `-O`. Явные уровни: `0` (без оптимизации; `__debug__` — истина), `1` (ассерты удаляются, `__debug__` — ложно) или `2` (строки документации также удаляются).

Функция вызывает [`SyntaxError`](#), если скомпилированный источник ошибочен и [`ValueError`](#), если источник содержит null байты.

Если вы хотите проанализировать код Python в его представлении AST, см. [`ast.parse\(\)`](#).

Вызывает событие аудита `compile` с аргументами `source` и `filename`. Событие также может вызываться неявной компиляцией.

---

**Примечание:** При компиляции строки с многострочным кодом в режиме `'single'` или `'eval'` ввод должен завершаться хотя бы одним символом новой строки. Это сделано для облегчения обнаружения неполных и полных операторов в модуле [`code`](#).

---

**Предупреждение:** Возможен сбой интерпретатора Python с достаточно большой/сложной строкой при компиляции в AST объект из-за ограничений глубины стека в компиляторе Python AST.

Изменено в версии 3.2: Разрешено использование символов новой строки Windows и Mac. Также ввод в режиме `'exec'` больше не должен заканчиваться новой строкой. Добавлен параметр `optimize`.

Изменено в версии 3.5: Ранее `TypeError` вызывался при обнаружении нулевых байтов в `source`.

Добавлено в версии 3.8: `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT` теперь можно передавать во флагах, чтобы включить высокоуровневую поддержку `await`, `async for` и `async with`.

**class** `complex`(`[real[, imag]]`)

Возвращает комплексное число со значением `real + imag*1j` или преобразовывает строку или число в комплексное число. Если первый параметр является строкой, он будет интерпретироваться как комплексное число, и функция должна вызываться без второго параметра. Второй параметр никогда не может быть строкой. Каждый аргумент может быть любого числового типа (включая комплексный). Если `imag` пропущен, он по умолчанию равен нулю, а конструктор выполняет числовое преобразование, например `int` и `float`. Если оба аргумента пропущены, возвращает `0j`.

Для общего Python объекта `x`, `complex(x)` делегирует `x.__complex__()`. Если `__complex__()` не определён, он возвращается к `__float__()`. Если `__float__()` не определён, он возвращается к `__index__()`.

---

**Примечание:** При преобразовании из строки строка не должна содержать пробелов вокруг центрального оператора `+` или `-`. Например, `complex('1+2j')` подходит, но `complex('1 + 2j')` поднимает `ValueError`.

---

Сложный тип описан в *Числовые типы — int, float, complex*.

Изменено в версии 3.6: Допускается группировка цифр с подчеркиванием, как в литералах кода.

Изменено в версии 3.8: Возвращается к `__index__()`, если `__complex__()` и `__float__()` не определены.

**delattr**(`object, name`)

Родственник `setattr()`. Аргументы — объект и строка. Строка должна быть именем одного из атрибутов объекта. Функция удаляет именованный атрибут, если это позволяет объект. Например, `delattr(x, 'foobar')` эквивалентен `del x.foobar`.

**class** `dict`(`**kwarg`)

**class** `dict`(`mapping, **kwarg`)

**class** `dict`(`iterable, **kwarg`)

Создать новый словарь. Объект `dict` — класс словаря. См. документацию по этому классу в `dict` и *Типы сопоставления — dict*.

Для других контейнеров см. встроенные классы `list`, `set` и `tuple`, а также модуль `collections`.

**dir**(`[object]`)

Без аргументов возвращает список имён в текущей локальной области. С аргументом попытается вернуть список допустимых атрибутов для данного объекта.

Если у объекта есть метод с именем `__dir__()`, этот метод будет вызван и должен вернуть список атрибутов. Это позволяет объектам, реализующим пользовательскую функцию `__getattr__()` или `__getattribute__()`, настраивать способ сообщения `dir()` своих атрибутов.

Если объект не предоставляет `__dir__()`, функция из всех сил пытается собрать информацию из атрибута `__dict__` объекта, если он определён, и из объекта его типа. Результирующий список не обязательно является полным и может быть неточным, если у объекта есть настраиваемый `__getattr__()`.

Механизм `dir()` по умолчанию ведёт себя по-разному с разными типами объектов, поскольку он пытается произвести наиболее подходящую, а не полную информацию:

- Если объект является объектом модуля, список содержит имена атрибутов модуля.
- Если объект является типе объектом или класса, список содержит имена его атрибутов и рекурсивно атрибутов его предков.
- В противном случае список содержит имена атрибутов объекта, имена атрибутов его класса и рекурсивно атрибуты родительских классов его класса.

Полученный список отсортирован по алфавиту. Например:

```
>>> import struct
>>> dir() # отображение имён в пространстве имён модулей # doctest:
↳+SKIP
['__builtins__', '__name__', 'struct']
>>> dir(struct) # показать имена в модуле struct # doctest: +SKIP
['Struct', '__all__', '__builtins__', '__cached__', '__doc__', '__file__',
 '__initializing__', '__loader__', '__name__', '__package__',
 '__clearcache', 'calcsize', 'error', 'pack', 'pack_into',
 'unpack', 'unpack_from']
>>> class Shape:
...     def __dir__(self):
...         return ['area', 'perimeter', 'location']
>>> s = Shape()
>>> dir(s)
['area', 'location', 'perimeter']
```

**Примечание:** Поскольку `dir()` предоставляется в первую очередь для удобства использования в интерактивном приглашении, она пытается предоставить доскональный или постоянное множество имён, и её подробное поведение может меняться в разных релизах. Например, атрибуты метакласса отсутствуют в списке результатов, если аргументом является класс.

### `divmod(a, b)`

Принять два (не комплексных) числа в качестве аргументов и вернуть пару чисел, состоящую из их частного и остатка при использовании целочисленного деления. К смешанным типам операндов применяются правила для бинарных арифметических операторов. Для целых чисел результат такой же, как  $(a // b, a \% b)$ . Для чисел с плавающей запятой результатом будет  $(q, a \% b)$ , где  $q$  обычно равно  $\text{math.floor}(a / b)$ , но может быть на 1 меньше. В любом случае  $q * b + a \% b$  очень близок к  $a$ , если  $a \% b$  отличен от нуля, у него тот же знак, что и  $b$  и  $0 \leq \text{abs}(a \% b) < \text{abs}(b)$ .

### `enumerate(iterable, start=0)`

Возвращает перечисляемый объект. `iterable` должен быть последовательностью, *итератором* или каким-либо другим объектом, поддерживающим итерацию. Метод `__next__()` итератора, возвращенный `enumerate()`, возвращает кортеж, содержащий счётчик (`start`, который по умолчанию равен 0) и значения, полученные в результате итерации по `iterable`.