**Python**[[комм 1]](https://ru.wikipedia.org/wiki/Python#cite_note-11) ([МФА](https://ru.wikipedia.org/wiki/%D0%9C%D0%A4%D0%90): [[ˈpʌɪθ(ə)n]](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%B4%D1%83%D0%BD%D0%B0%D1%80%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D1%84%D0%BE%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D1%84%D0%B0%D0%B2%D0%B8%D1%82); в русском языке распространено название *пито́н*[[11]](https://ru.wikipedia.org/wiki/Python#cite_note-12)) — [высокоуровневый язык программирования](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. [Синтаксис](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) ядра Python минималистичен. В то же время [стандартная библиотека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_Python) включает большой объём полезных функций.

Python поддерживает [структурное](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [объектно-ориентированное](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [функциональное](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [императивное](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и [аспектно-ориентированное программирование](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Основные архитектурные черты — [динамическая типизация](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), [автоматическое управление памятью](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), полная [интроспекция](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D1%80%D0%BE%D1%81%D0%BF%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), механизм [обработки исключений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), поддержка [многопоточных вычислений](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C), высокоуровневые [структуры данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Поддерживается разбиение программ на [модули](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), которые, в свою очередь, могут объединяться в пакеты.

Эталонной реализацией Python является интерпретатор [CPython](https://ru.wikipedia.org/wiki/CPython), поддерживающий большинство активно используемых платформ[[12]](https://ru.wikipedia.org/wiki/Python#cite_note-13). Он распространяется под [свободной лицензией](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая [проприетарные](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BF%D1%80%D0%B8%D0%B5%D1%82%D0%B0%D1%80%D0%BD%D0%BE%D0%B5_%D0%9F%D0%9E)[[13]](https://ru.wikipedia.org/wiki/Python#cite_note-14). Есть [реализация интерпретатора для JVM](https://ru.wikipedia.org/wiki/Jython) с возможностью [компиляции](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80), [CLR](https://ru.wikipedia.org/wiki/IronPython), [LLVM](https://ru.wikipedia.org/wiki/LLVM), другие независимые реализации. Проект [PyPy](https://ru.wikipedia.org/wiki/PyPy) использует [JIT-компиляцию](https://ru.wikipedia.org/wiki/JIT-%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%86%D0%B8%D1%8F), которая значительно увеличивает скорость выполнения Python-программ.

Python — активно развивающийся [язык программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года. Язык не подвергался официальной стандартизации, роль стандарта де-факто выполняет [CPython](https://ru.wikipedia.org/wiki/CPython), разрабатываемый под контролем автора языка. В настоящий момент Python занимает третье место в рейтинге TIOBE с показателем 8,5 %. Аналитики отмечают, что это самый высокий балл Python за все время его присутствия в рейтинге.

**Операторы**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=7) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=7)]

Набор операторов достаточно традиционен.

* Условный оператор if (если). Альтернативный блок после else (иначе). Если условий и альтернатив несколько, можно использовать elif (сокр. от else if).
* Операторы цикла while (пока) и for (для). Внутри цикла возможно применение break и continue для прерывания цикла и перехода сразу к следующей итерации, соответственно.
* Оператор определения класса class.
* Оператор определения функции, метода или генератора def. Внутри возможно применение return (возврат) для возврата из функции или метода, а в случае генератора — yield (давать).
* Оператор обработки исключений try — except — else или try — finally (начиная с версии 2.5, можно использовать finally, except и else в одном блоке).
* Оператор pass ничего не делает. Используется для пустых блоков кода.

Одной из интересных синтаксических особенностей языка является выделение [блоков](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) кода с помощью отступов (пробелов или табуляций), поэтому в Python отсутствуют [операторные скобки](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D1%8B%D0%B5_%D1%81%D0%BA%D0%BE%D0%B1%D0%BA%D0%B8) begin/end, как в языке [Паскаль](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D1%8C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), или фигурные скобки, как в [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). Такой «трюк» позволяет сократить количество строк и символов в программе и приучает к «хорошему» стилю программирования. С другой стороны, поведение и даже корректность программы может зависеть от начальных пробелов в тексте. Некоторым такое поведение может показаться неинтуитивным и неудобным.

**Выражения**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=8) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=8)]

Выражение является полноправным оператором в Python. Состав, синтаксис, ассоциативность и приоритет операций достаточно привычны для языков программирования и призваны минимизировать употребление скобок.

Отдельно стоит упомянуть *операцию форматирования* для строк (работает по аналогии с printf() из Си), которая использует тот же символ, что и взятие остатка от деления:

>>> str\_var = "world"

>>> **print**("Hello, **%s**" % str\_var)

Hello, world

Python имеет удобные *цепочечные сравнения*. Такие условия в программах — не редкость:

1 <= a < 10 **and** 1 <= b < 20

Кроме того, логические операции (or и and) являются [ленивыми](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BD%D0%B8%D0%B2%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F): если для вычисления значения операции достаточно первого операнда, этот операнд и является результатом, в противном случае вычисляется второй операнд логической операции. Это основывается на свойствах [алгебры логики](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0_%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B8): например, если один аргумент операции «ИЛИ» (or) является истиной, то и результат этой операции всегда является истиной. В случае, если второй операнд является сложным выражением, это позволяет сократить издержки на его вычисление. Этот факт широко использовался до версии 2.5 вместо условной конструкции:

a < b **and** "меньше" **or** "больше или равно"

Встроенные типы данных, как правило, имеют особый синтаксис для своих литералов (записанных в исходном коде констант):

"строка и Юникод-строка одновременно"

'строка и Юникод-строка одновременно'

*"""тоже строка и Юникод-строка одновременно"""*

True **or** False *# булевы литералы*

3.14 *# число с плавающей запятой*

0b1010 + 0o12 + 0xA *# числа в двоичной, восьмеричной и шестнадцатеричной системах счисления*

1 + 2j *# комплексное число*

[1, 2, "a"] *# список*

(1, 2, "a") *# кортеж*

{'a': 1, 'b': 'B'} *# словарь*

{'a', 6, 8.8} *# множество*

**lambda** x: x\*\*2 *# анонимная функция*

Для списков (и других последовательностей) Python предлагает набор операций над срезами. Особенностью является индексация, которая может показаться новичку странной, но раскрывает свою согласованность по мере использования. Индексы элементов списка начинаются с нуля. Запись среза s[N:M] означает, что в срез попадают все элементы от N включительно до M не включая. В качестве иллюстрации можно посмотреть [пример работы с последовательностями](https://ru.wikiversity.org/wiki/%D0%9F%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D1%8B_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC_%D0%BD%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5_Python#Примеры_работы_с_последовательностями). При этом индекс можно не указывать. Например, запись s[:M] означает, что в срез попадают все элементы с самого начала; запись s[N:] означает, что попадают все элементы до конца среза; запись s[:] означает, что попадают все элементы с начала и до конца.

**Имена**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=9) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=9)]

Имя (идентификатор) может начинаться с латинской буквы (в Python 3 — буквы любого алфавита в Юникоде, например кириллицы) любого регистра или подчёркивания, после чего в имени можно использовать и цифры. В качестве имени нельзя использовать ключевые слова (их список можно узнать по import keyword; print(keyword.kwlist)) и нежелательно переопределять встроенные имена. Имена, начинающиеся с символа подчёркивания, имеют специальное значение[[28]](https://ru.wikipedia.org/wiki/Python#cite_note-32).

В каждой точке программы интерпретатор имеет доступ к трём [пространствам имён](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%B8%D0%BC%D1%91%D0%BD_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (то есть отображениям имён в объекты): локальному, глобальному и встроенному.

*Области видимости* имён могут быть вложенными друг в друга (внутри определяемой функции видны имена из окружающего блока кода). На практике с областями видимости и связыванием имён связано несколько правил «хорошего тона», о которых можно подробнее узнать из документации.

**Строки документации**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=10) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=10)]

Python предлагает механизм [документирования](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F) кода pydoc. В начало каждого модуля, класса, функции вставляется строка документации — [*docstring*](https://ru.wikipedia.org/w/index.php?title=Docstring&action=edit&redlink=1)*(*[*англ.*](https://en.wikipedia.org/wiki/docstring)*)*. Строки документации остаются в коде на момент времени исполнения, и в язык встроен доступ к документации[[29]](https://ru.wikipedia.org/wiki/Python#cite_note-33)(переменная \_\_doc\_\_), что используется современными IDE ([Интегрированная среда разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)) (например, [Eclipse](https://ru.wikipedia.org/wiki/Eclipse_(%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8))).

В интерактивном режиме можно получить помощь, сгенерировать гипертекстовую документацию по целому модулю или даже применить [*doctest*](https://ru.wikipedia.org/w/index.php?title=Doctest&action=edit&redlink=1)*(*[*англ.*](https://en.wikipedia.org/wiki/doctest)*)* для автоматического тестирования модуля.

**Директивы**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=11) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=11)]

Начиная с Python 2.3, для использования в тексте программы символов, не входящих в ASCII, необходимо явно указывать кодировку исходного кода в начале модуля, например:

*# -\*- coding: utf-8 -\*-*

*# или*

*# coding: utf-8*

После этого можно использовать, например, кириллицу в Unicode-литералах. Но на самом деле даже если написать:

*# coding: utf*

то Python «поймёт», что вы хотели сделать.

**Обработка исключений**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=17) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=17)]

[Обработка исключений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9) поддерживается в Python посредством операторов try, except, else, finally, raise, образующих блок обработки исключения. В общем случае блок выглядит следующим образом:

**try**:

*# Здесь код, который может вызвать исключение*

**raise** **Exception**("message") *# Exception, это один из стандартных типов исключения (всего лишь класс),*

*# может использоваться любой другой, в том числе свой*

**except** (Тип исключения1, Тип исключения2, …) **as** Переменная:

*# Код в блоке выполняется, если тип исключения совпадает с одним из типов*

*# (Тип исключения1, Тип исключения2, …) или является наследником одного*

*# из этих типов.*

*# Полученное исключение доступно в необязательной Переменной.*

**except** (Тип исключения3, Тип исключения4, …) **as** Переменная:

*# Количество блоков except не ограничено*

**raise** *# Сгенерировать исключение "поверх" полученного; без параметров - повторно сгенерировать полученное*

**except**:

*# Будет выполнено при любом исключении, не обработанном типизированными блоками except*

**else**:

*# Код блока выполняется, если не было поймано исключений.*

**finally**:

*# Будет исполнено в любом случае, возможно после соответствующего*

*# блока except или else*

Совместное использование else, except и finally стало возможно только начиная с Python 2.5. Информация о текущем исключении всегда доступна через sys.exc\_info(). Кроме значения исключения, Python также сохраняет состояние стека вплоть до точки возбуждения исключения — так называемый traceback.

В отличие от компилируемых языков программирования, в Python использование исключения не приводит к значительным накладным расходам (а зачастую даже позволяет ускорить исполнение программ) и очень широко используется. Исключения согласуются с философией Python (10-й пункт «[дзена Python](https://ru.wikipedia.org/wiki/Python#Философия)» — «Ошибки никогда не должны умалчиваться») и являются одним из средств поддержки «[утиной типизации](https://ru.wikipedia.org/wiki/%D0%A3%D1%82%D0%B8%D0%BD%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F)».

Иногда вместо явной обработки исключений удобнее использовать блок [with](https://ru.wikipedia.org/wiki/Python#Управление_контекстом_выполнения) (доступен, начиная с Python 2.5).

**Итераторы**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=18) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=18)]

В программах на Python широко используются [итераторы](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Цикл for может работать как с последовательностью, так и с итератором. Большинство коллекций предоставляют итераторы, итераторы могут также определяться пользователем для собственных объектов. Модуль itertools стандартной библиотеки содержит средства работы с итераторами.

**Генераторы**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=19) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=19)]

Одной из интересных возможностей языка являются **генераторы** — функции, сохраняющие внутреннее состояние: значения локальных переменных и текущую инструкцию (см. также: [сопрограммы](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)). Генераторы могут использоваться как [итераторы](https://ru.wikipedia.org/wiki/%D0%98%D1%82%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) для структур данных и для [ленивых вычислений](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BD%D0%B8%D0%B2%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F). См. пример: [генератор чисел Фибоначчи](https://ru.wikiversity.org/wiki/%D0%9F%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D1%8B_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC_%D0%BD%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5_Python#Генератор_чисел_Фибоначчи).

При вызове генератора функция немедленно возвращает объект-итератор, который хранит текущую точку исполнения и состояние локальных переменных функции. При запросе следующего значения (посредством метода next(), неявно вызываемого в цикле for) генератор продолжает исполнение функции от предыдущей точки останова до следующего оператора yield или return.

В Python 2.4 появились **генераторные выражения** — выражения, дающие в результате генератор. Генераторные выражения позволяют сэкономить память там, где иначе требовалось бы использовать список с промежуточными результатами:

>>> sum(i **for** i **in** xrange(1, 100) **if** i % 2 != 0)

2500

В этом примере суммируются все нечётные числа от 1 до 99.

Начиная с версии 2.5, Python поддерживает полноценные сопроцедуры: теперь в генератор можно передавать значения с помощью метода send() и возбуждать в его контексте исключения с помощью метода throw().

Также Python поддерживает вложенные генераторы. Например, для создания двумерного массива нужно разместить генератор списка, являющегося строкой, внутри генератора всех строк: [[0 for j in range(m)] for i in range(n)]

**Управление контекстом выполнения**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=20) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=20)]

В Python 2.5 появились средства для управления контекстом выполнения блока кода — оператор with и модуль contextlib. См.: [пример](https://ru.wikiversity.org/wiki/%D0%9F%D1%80%D0%B8%D0%BC%D0%B5%D1%80%D1%8B_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC_%D0%BD%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B5_Python#Управление_контекстом_выполнения).

Оператор может применяться в тех случаях, когда до и после некоторых действий должны обязательно выполняться некоторые другие действия, независимо от возбуждённых в блоке исключений или операторов return: файлы должны быть закрыты, ресурсы освобождены, перенаправление стандартного ввода вывода закончено и т. п. Оператор улучшает читаемость кода, а значит, помогает предотвращать ошибки.

**Декораторы**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=21) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=21)]

Начиная с версии 2.4, Python позволяет использовать так называемые *декораторы*[[30]](https://ru.wikipedia.org/wiki/Python#cite_note-34)[[31]](https://ru.wikipedia.org/wiki/Python#cite_note-35) (не следует путать с [одноимённым](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BA%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) [шаблоном проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) для поддержки существующей практики преобразования функций и методов в месте определения (декораторов может быть несколько). Для декораторов используется символ @ в строках, предшествующих определению функции или метода. Синтаксис декорирования является [синтаксическим сахаром](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D1%81%D0%B0%D1%85%D0%B0%D1%80)[[32]](https://ru.wikipedia.org/wiki/Python#cite_note-36)[[33]](https://ru.wikipedia.org/wiki/Python#cite_note-37) для удобочитаемости: код

@staticmethod

**def** my\_wonderful\_method():

**return** "Некоторый метод"

полностью эквивалентен

**def** my\_wonderful\_method():

**return** "Некоторый метод"

my\_wonderful\_method = staticmethod(my\_wonderful\_method)

Сам декоратор является функцией, получающей в качестве первого аргумента декорируемую функцию. Для передачи дополнительных аргументов можно использовать синтаксис @декоратор(аргументы). Декораторы можно считать элементом [аспектно-ориентированного программирования](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D0%BF%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

С версии 2.6 декораторы можно использовать с классами аналогично функциям.

**Регулярные выражения**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=22) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=22)]

Формат [регулярных выражений](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F) унаследован из [Perl](https://ru.wikipedia.org/wiki/Perl) с некоторыми отличиями. Для их использования требуется импортировать модуль re[[34]](https://ru.wikipedia.org/wiki/Python#cite_note-38), являющийся частью стандартной библиотеки.

**Другие возможности**[[править](https://ru.wikipedia.org/w/index.php?title=Python&veaction=edit&section=23) | [править код](https://ru.wikipedia.org/w/index.php?title=Python&action=edit&section=23)]

В Python есть ещё несколько возможностей, отличающих его от многих других языков высокой гибкостью и динамичностью.

Например, класс является объектом, а в операторе определения класса можно использовать выражения в списке родительских классов.

**def** get\_class():

**return** dict

**class** **D**(get\_class()):

**pass**

d = D()

Можно модифицировать многие объекты во время исполнения, например классы:

>>> **class** **X**(object): **pass**

…

>>> y = X()

>>> y.wrong\_method() *# такого метода пока нет*

Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>

**AttributeError**: 'X' object has no attribute 'wrong\_method'

>>> X.wrong\_method = **lambda** self : 'im here' *# добавим его*

>>> y.wrong\_method() *# так как доступ к методу приводит к поиску по \_\_dict\_\_ класса,*

'im here' *# то wrong\_method становится доступным всем экземплярам*