Встроенные функции

**abs()** - возвращает абсолютное значение числа. Аргументом может быть целое число, число с плавающей запятой или реализация объекта \_\_abs\_\_(). Если аргументом является комплексное число, возвращается его величина.

**aiter()** - возвращает асинхронный итератор для асинхронной итерации. Эквивалентно вызову x.\_\_aiter\_\_().

**all()** - возврат True если все элементы итерируемого имеют значение true (или если итерируемый пуст).

**anext()** - когда ожидается, верните следующий элемент из заданного асинхронный итератор, или По умолчанию если задано и итератор исчерпан. Это асинхронный вариант next() встроенных функций и ведет себя аналогично. Это вызывает \_\_anext\_\_() метод async\_iterator, возвращающий ожидаемый. Ожидание этого возвращает следующее значение итератора. Если По умолчанию задано, оно возвращается, если итератор исчерпан, в противном случае StopAsyncIteration вызывается.

**any()** - возврат True если какой-либо элемент из итерируемого имеет значение true. Если итерируемый пуст, верните False.

**ascii()** - как repr(), возвращает строку, содержащую печатное представление объекта, но экранирует символы, отличные от ASCII, в строке, возвращаемой repr() с помощью \x, \u, или \U экранирует. Это генерирует строку, аналогичную той, что возвращена repr() в Python 2.

**bin()** - преобразуйте целое число в двоичную строку с префиксом “0b”. Результатом является допустимое выражение Python. Если x не является Python int объект, он должен определять \_\_index\_\_() метод, возвращающий целое число.

**bool()** - возвращает логическое значение, то есть одно из True или False. x преобразуется с использованием стандартной процедуры проверки на истинность. Если x значение false или пропущено, это возвращает False; в противном случае оно возвращает True. boolКласс является подклассом int (см. Числовые типы — int, float, complex). Его нельзя далее подразделять на подклассы. Его единственными экземплярами являются False и True (см. Логические значения).

**breakpoint()** - эта функция отправляет вас в отладчик на сайте вызова. В частности, она вызывает sys.breakpointhook(), минуя args и kws непосредственно через. По умолчанию sys.breakpointhook() вызовы pdb.set\_trace() не ожидают никаких аргументов. В данном случае это чисто удобная функция, поэтому вам не нужно явно импортировать pdb или вводить столько кода для ввода в отладчик. Однако sys.breakpointhook() может быть установлена какая-либо другая функция и breakpoint() автоматически вызовет ее, что позволит вам перейти к выбранному отладчику. Если sys.breakpointhook() недоступна, эта функция вызовет RuntimeError.

**bytearray() -** возвращает новый массив байтов. bytearray Класс представляет собой изменяемую последовательность целых чисел в диапазоне 0 <= x < 256. Он содержит большинство обычных методов изменяемых последовательностей, описанных в разделе Типы изменяемых последовательностей, а также большинство методов, которые есть у bytes типа, см. Операции с байтами и Bytearray.

**bytes()** - возвращает новый объект “bytes”, который представляет собой неизменяемую последовательность целых чисел в диапазоне 0 <= x < 256. bytes является неизменяемой версией bytearray – она имеет те же методы, которые не изменяются, и то же поведение индексации и нарезки.

**callable()** - возвращает, True если аргумент object кажется вызываемым, False если нет. Если это возвращает True, все еще возможно, что вызов завершается неудачно, но, если это так False, вызов object никогда не будет успешным. Обратите внимание, что классы могут вызываться (вызов класса возвращает новый экземпляр); экземпляры могут вызываться, если их класс имеет \_\_call\_\_() метод.

**chr()** - возвращает строку, представляющую символ, кодовой точкой которого в Юникоде является целое число i. Например, chr(97) возвращает строку 'a', в то время как chr(8364) возвращает строку '€'. Это обратное ord().

**classmethod()** - преобразуйте метод в метод класса.

**compile()** - скомпилируйте исходный код в код или объект AST. Объекты кода могут выполняться с помощью exec() или eval(). источником может быть обычная строка, строка байтов или объект AST. Обратитесь к ast документации модуля для получения информации о том, как работать с объектами AST.

**complex()** - возвращает комплексное число со значением real + imag\*1j или преобразует 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 опущен, по умолчанию он равен нулю, и конструктор служит для числового преобразования, подобного int и float. Если оба аргумента опущены, возвращает 0j.

**delattr()** — это родственник setattr(). Аргументами являются объект и строка. Строка должна быть именем одного из атрибутов объекта. Функция удаляет именованный атрибут при условии, что объект это разрешает. Например, delattr(x, 'foobar') эквивалентно del x.foobar. имя не обязательно должно быть идентификатором Python (см. setattr()).

**dict()** - создайте новый словарь. dict Объектом является класс dictionary. Документацию об этом классе см. в разделе dict и Mapping Types — dict.

**dir()** - без аргументов верните список имен в текущей локальной области видимости. С помощью аргумента попытайтесь вернуть список допустимых атрибутов для этого объекта.

**divmod() -** принимают два (не сложных) числа в качестве аргументов и возвращают пару чисел, состоящих из их частного и остатка при использовании целочисленного деления. При смешанных типах операндов применяются правила для двоичных арифметических операторов. Для целых чисел результат такой же, как (a // b, a % b). Для чисел с плавающей запятой результатом является (q, a % b), где q обычно math.floor(a / b) равно, но может быть на 1 меньше. В любом случае q \* b + a % b очень близко к a, если a % b не равно нулю, то оно имеет тот же знак, что и b, и 0 <= abs(a % b) < abs(b).

**enumerate()** - возвращает объект enumerate. iterable должна быть последовательностью, итератором или каким-либо другим объектом, поддерживающим итерацию. \_\_next\_\_()Метод итератора, возвращаемый enumerate(), возвращает кортеж, содержащий количество (с начала, которое по умолчанию равно 0) и значения, полученные в результате перебора iterable.

**eval()** - аргументами являются строка и необязательные глобальные и локальные значения. Если они указаны, глобальными должны быть словари. Если указаны, локальными могут быть любые объекты сопоставления.

**exec() -** эта функция поддерживает динамическое выполнение кода на Python. объект должен быть либо строкой, либо объектом кода. Если это строка, то строка анализируется как набор инструкций Python, который затем выполняется (если не возникает синтаксическая ошибка). 1 Если это объект code, он просто выполняется. Ожидается, что во всех случаях выполняемый код будет действителен как файловый ввод (см. раздел Файловый ввод в Справочном руководстве). Имейте в виду, что операторы nonlocal, yield и return не могут использоваться вне определений функций даже в контексте кода, передаваемого в exec() функцию. Возвращаемое значение равно None.

**filter()** - создайте итератор из тех элементов iterable, для которых функция имеет значение true. iterable может быть либо последовательностью, контейнером, поддерживающим итерацию, либо итератором. Если функция имеет значение None, предполагается, что функция идентификации, то есть все элементы iterable, которые являются false, удаляются.

**float() -** возвращает число с плавающей запятой, построенное из числа или строки x.

**format()** - преобразуйте значение в “форматированное” представление, управляемое format\_spec. Интерпретация format\_spec будет зависеть от типа аргумента value; однако существует стандартный синтаксис форматирования, который используется большинством встроенных типов: мини-язык спецификации формата.

**frozenset()** - возвращает новый frozenset объект, необязательно с элементами, взятыми из iterable. frozenset является встроенным классом. Документацию об этом классе см. в разделах frozenset и Set Types — set, frozenset.

**getattr() -** возвращает значение именованного атрибута объекта. name должно быть строкой. Если строка является именем одного из атрибутов объекта, результатом является значение этого атрибута. Например, getattr(x, 'foobar') эквивалентно x.foobar. Если именованный атрибут не существует, возвращается значение по умолчанию, если оно указано, в противном случае AttributeError возникает вопрос. имя не обязательно должно быть идентификатором Python (см. setattr()).

**globals()** - возвращает словарь, реализующий текущее пространство имен модуля. Для кода внутри функций это значение устанавливается при определении функции и остается неизменным независимо от того, где вызывается функция.

**hasattr()** - аргументами являются объект и строка. Результатом будет, True если строка является именем одного из атрибутов объекта, False если нет. (Это реализуется путем вызова getattr(object, name) и просмотра, вызывает ли это AttributeError или нет.)

**hash()** - возвращает хэш-значение объекта (если оно у него есть). Хэш-значения представляют собой целые числа. Они используются для быстрого сравнения ключей словаря во время поиска по словарю. Числовые значения, которые сравниваются равными, имеют одинаковое хэш-значение (даже если они разных типов, как в случае с 1 и 1.0).

**help()** - вызовите встроенную справочную систему. (Эта функция предназначена для интерактивного использования.) Если аргумент не указан, интерактивная справочная система запускается на консоли интерпретатора. Если аргументом является строка, то строка просматривается как имя модуля, функции, класса, метода, ключевого слова или раздела документации, и на консоли выводится страница справки. Если аргументом является объект любого другого типа, создается страница справки по объекту.

**hex()** - преобразуйте целое число в шестнадцатеричную строку в нижнем регистре с префиксом “0x”. Если x не является int объектом Python, он должен определить \_\_index\_\_() метод, который возвращает целое число. Некоторые примеры:

**id()** - возвращают “идентификатор” объекта. Это целое число, которое гарантированно будет уникальным и постоянным для этого объекта в течение его жизненного цикла. Два объекта с неперекрывающимися временами жизни могут иметь одинаковое id() значение.

**input()** - если присутствует аргумент prompt, он записывается в стандартный вывод без перевода строки в конце. Затем функция считывает строку из входных данных, преобразует ее в строку (удаляя завершающий символ новой строки) и возвращает ее. Когда EOF считывается, вызывается EOFError. Пример:

**int()** - возвращает целочисленный объект, созданный из числа или строки x, или возвращает 0, если аргументы не указаны. Если x определяет \_\_int\_\_(), int(x) возвращает x.\_\_int\_\_(). Если x определяет \_\_index\_\_(), он возвращает x.\_\_index\_\_(). Если x определяет \_\_trunc\_\_(), он возвращает x.\_\_trunc\_\_(). Для чисел с плавающей запятой это значение усекается до нуля.

**isinstance()** - возвращает, True если аргумент object является экземпляром аргумента classinfo или его подкласса (прямого, косвенного или виртуального). Если object не является объектом данного типа, функция всегда возвращает False. Если classinfo представляет собой кортеж объектов типа (или, рекурсивно, другие подобные кортежи) или объединяющий тип нескольких типов, верните, True если object является экземпляром любого из типов. Если classinfo не является типом или кортежем типов и подобных кортежей, возникает TypeError исключение. TypeErrorможет не быть вызван для недопустимого типа, если предыдущая проверка прошла успешно.

**issubclass()** - возвращает, True если class является подклассом (прямым, косвенным или виртуальным) classinfo. Класс считается подклассом самого себя. classinfo может быть кортежем объектов класса (или, рекурсивно, другими подобными кортежами) или типом объединения, и в этом случае возвращается, True если class является подклассом любой записи в classinfo. В любом другом случае возникает TypeError исключение.

**iter()** - возвращает объект iterator. Первый аргумент интерпретируется очень по-разному в зависимости от наличия второго аргумента. Без второго аргумента object должен быть объектом коллекции, который поддерживает повторяющийся протокол (\_\_iter\_\_() метод), или он должен поддерживать протокол sequence (\_\_getitem\_\_() метод с целочисленными аргументами, начинающимися с 0). Если он не поддерживает ни один из этих протоколов, вызывается TypeError. Если задан второй аргумент, sentinel, то object должен быть вызываемым объектом. Созданный в этом случае итератор будет вызывать object без аргументов для каждого вызова своего \_\_next\_\_() метода; если возвращаемое значение равно sentinel, StopIteration будет вызвано, в противном случае значение будет возвращено.

**len()** - возвращает длину (количество элементов) объекта. Аргументом может быть последовательность (такая как строка, байты, кортеж, список или диапазон) или коллекция (такая как словарь, набор или замороженный набор).

**list()** - на самом деле это не функция, list а изменяемый тип последовательности, как описано в списках и типах последовательностей — list, tuple, range.

**locals()** - обновите и верните словарь, представляющий текущую локальную таблицу символов. Свободные переменные возвращаются с помощью locals(), когда она вызывается в функциональных блоках, но не в блоках классов. Обратите внимание, что на уровне модуля locals() и globals() являются одним и тем же словарем.

**map()** - возвращает итератор, который применяет функцию к каждому элементу iterable, выдавая результаты. Если передаются дополнительные аргументы iterables, функция должна принимать такое количество аргументов и применяется к элементам из всех iterables параллельно. При использовании нескольких итераций итератор останавливается, когда исчерпывается самая короткая итеративная функция. О случаях, когда входные данные функции уже упорядочены в кортежи аргументов, см. itertools.starmap().

**max()** - возвращает самый большой элемент в итерируемом или самый большой из двух или более аргументов.

**memoryview() -** возвращает объект “представление памяти”, созданный на основе заданного аргумента. Дополнительные сведения см. в разделе Представления памяти.

**min()** - возвращает наименьший элемент в итерируемом или наименьший из двух или более аргументов.

**next()** - извлеките следующий элемент из итератора, вызвав его \_\_next\_\_() метод. Если задано значение по умолчанию, оно возвращается, если итератор исчерпан, в противном случае StopIteration вызывается.

**object()** - возвращает новый безликий объект. object является базовым для всех классов. У него есть методы, общие для всех экземпляров классов Python. Эта функция не принимает никаких аргументов.

**oct()** - преобразуйте целое число в восьмеричную строку с префиксом “0o”. Результатом является допустимое выражение Python. Если x не является int объектом Python, он должен определить \_\_index\_\_() метод, который возвращает целое число. Например:

**open()** - откройте файл и верните соответствующий файловый объект. Если файл не может быть открыт, OSError выдается сообщение. Дополнительные примеры использования этой функции см. в разделе Чтение и запись файлов.

**ord()** - учитывая строку, представляющую один символ Юникода, верните целое число, представляющее кодовую точку Юникода этого символа. Например, ord('a') возвращает целое число 97, а ord('€') возвращает значение (знак Евро) 8364. Это значение, обратное chr().

**pow()** - верните base в power exp; если присутствует mod, верните base в power exp по модулю mod (вычисляется более эффективно, чем pow(base, exp) % mod). Форма с двумя аргументами pow(base, exp) эквивалентна использованию оператора power: base\*\*exp.

**print()** - распечатайте объекты в файл текстового потока, разделенные sep и сопровождаемые end. sep, end, file и flush, если они присутствуют, должны быть указаны в качестве аргументов ключевого слова.

**property()** - возвращает атрибут property.

**range()** - на самом деле это не функция, range а неизменяемый тип последовательности, как описано в диапазонах и типах последовательностей — список, кортеж, диапазон.

**repr()** - возвращает строку, содержащую печатное представление объекта. Для многих типов эта функция пытается вернуть строку, которая при передаче в eval() выдает объект с тем же значением; в противном случае представление представляет собой строку, заключенную в угловые скобки, которая содержит имя типа объекта вместе с дополнительной информацией, часто включающей имя и адрес объекта. Класс может управлять тем, что эта функция возвращает для своих экземпляров, определяя \_\_repr\_\_() метод. Если sys.displayhook() он недоступен, эта функция вызовет RuntimeError.

**reversed()** - возвращает обратный итератор. seq должен быть объектом, который имеет \_\_reversed\_\_() метод или поддерживает протокол sequence (\_\_len\_\_() метод и \_\_getitem\_\_() метод с целочисленными аргументами, начинающимися с 0).

**round()** - возвращает число, округленное с точностью до ndigits после запятой. Если ndigits опущен или есть None, он возвращает ближайшее целое число к своим входным данным.

**set()** - возвращает новый set объект, необязательно с элементами, взятыми из iterable. set является встроенным классом. Документацию об этом классе см. в разделах set и Set Types — set, frozenset.

**setattr()** **— это** аналог getattr(). Аргументами являются объект, строка и произвольное значение. Строка может называть существующий атрибут или новый атрибут. Функция присваивает значение атрибуту при условии, что объект это допускает. Например, setattr(x, 'foobar', 123) эквивалентно x.foobar = 123.

**slice()** - возвращает объект slice, представляющий набор индексов, указанных в range(start, stop, step). Аргументы start и step по умолчанию имеют значение None. Объекты Slice имеют атрибуты данных, доступные только для чтения start, stop и step, которые просто возвращают значения аргументов (или их значения по умолчанию). У них нет другой явной функциональности; однако они используются NumPy и другими сторонними пакетами. Объекты Slice также генерируются при использовании расширенного синтаксиса индексирования. Например: a[start:stop:step] или a[start:stop, i]. Смотрите itertools.islice() альтернативную версию, которая возвращает итератор.

**sorted()** - возвращает новый отсортированный список из элементов в iterable.

**staticmethod()** - преобразуйте метод в статический метод.

**str()** - возвращает str версию объекта.

**sum()** - суммирует начало и элементы итерируемого объекта слева направо и возвращает итоговое значение. Элементами iterable обычно являются числа, а начальное значение не может быть строкой.

**super()** - возвращает прокси-объект, который делегирует вызовы методов родительскому классу типа. Это полезно для доступа к унаследованным методам, которые были переопределены в классе.

**tuple()** - на самом деле это не функция, tuple а неизменяемый тип последовательности, как описано в кортежах и типах последовательностей — list, tuple, range.

**type()** - с помощью одного аргумента возвращайте тип объекта. Возвращаемое значение представляет собой объект типа и, как правило, тот же объект, что и возвращаемый object.\_\_class\_\_.

**vars()** - возвращает \_\_dict\_\_ атрибут для модуля, класса, экземпляра или любого другого объекта с \_\_dict\_\_ атрибутом.

**zip()** - выполняйте параллельные переборы нескольких итераций, создавая кортежи с элементом из каждой.

**\_\_import\_\_()** - эта функция вызывается с помощью import инструкции. Его можно заменить (импортировав builtins модуль и присвоив ему builtins.\_\_import\_\_), чтобы изменить семантику import инструкции, но делать это настоятельно не рекомендуется, поскольку обычно проще использовать перехваты импорта (см. PEP 302) для достижения тех же целей и не вызывает проблем с кодом, который предполагает, что используется реализация импорта по умолчанию. Прямое использование \_\_import\_\_() также не рекомендуется в пользу importlib.import\_module().