Полезные функци

|  |  |  |
| --- | --- | --- |
| **input()** | Input(“Text”)  Import sys  age = int(sys.stdin.readline()) | Аналогично функции input. Ручной ввод данных |
| **abs()** | abs(a) | Модуль числа |
| **dir()** | dir(['это', 'короткий', 'список'])  Вернет: ['\_\_add\_\_', '\_\_class\_\_','append', 'count',……]  dir(“имя модуля”) | Возвращает все возможные действия с обектом  Вернет список всех атрибутов, определен­ных внутри модуля. |
| **help()** | help(popcorn.upper) | Справка по функции для обекта |
| **eval()** | eval('print("вау!")')  calc = input('Введите выражение: ')  Введите выражение: 12\*52  eval(calc) | Функция eval принимает в качестве аргумента строку и выполняет ее, считая, что это код на языке Python. |
| [‘[1, 2, 3]’, “{‘a’: 1, ‘b’: 2}\n”]  >>> **eval(parts[0])**  [1, 2, 3] | *# Преобразовать строку в списке в объект* |
| >>> **objects = [eval(P) for P in parts]**  [[1, 2, 3], {‘a’: 1, ‘b’: 2}] | *# То же самое для всех строк в списке. Получаем список с вложенным списком и словарем* |
| **exec()** | Str\_py = '''print('бутерброд') print('с колбасой')'''  exec(Str\_py) | Функция exec похожа на eval, но с ее помощью можно выполнять более сложный код. exec, в отличие от eval, не возвращает значения. Функция exec может пригодиться для запуска небольших программ, которые главная Python-программа считывает из файлов, то есть для выполнения программ внутри другой программы |
| **int()** | int(123.456)  >>>123 | int служит для преобразования строки или числа в целое число, при этом знаки после запятой (если они есть) отбрасываются. При попытке воспользоваться функцией int для преобразования **строки** с **вещественным** числом в целое вы получите ошибку |
| **len()** | len('это тестовая строка')  >>>19 | len возвращает «длину» объекта, то есть количество элементов в нем. |
| **max()**  **min()** | max(10, 300, 450, 50, 90)  min(10, 300, 450, 50, 90) | Max/min возвращает наибольший/наименьший элемент списка, кортежа или строки. |
| **range()** | range(0, 5) | Генератор последовательности. От и до с шагом |
| **sum()** | sum(my\_list\_of\_numbers)  sum(10, 20, 30) | sum складывает элементы списка, возвращая их сумму |
| **print()** | Print(“Text1”,”Text2”,end=””, sep=”:”)  print([object, **...**][, sep=’ ‘][, end=’\n’][, file=sys.stdout])  print(x, y, z, sep=’...’, file=open(‘data.txt’, ‘w’))  import sys  **sys.stdout = open(‘log.txt’, ‘a’)**  *# Перенаправить вывод в файл*  **Print(x, y, x)**  *# Текст появится в файле log.txt*  **temp = sys.stdout** *# Сохранить для последующего восстановления*  **sys.stdout = open(‘log.txt’, ‘a’)**  **sys.stdout.close()**  *# Вытолкнуть буферы на диск*  **sys.stdout = temp**  *или* ***sys.stdout= sys.\_\_stdout\_\_***  *# Восстановить первоначальный поток* | **Sep –**разделитель между выводимыми значениями.  **end –** строка, добавляемая в конец выводимого текста. По умолчанию содер­жит символ конца строки \n. Если в этом аргументе передать пустую стро­ку, следующий вызов функции print начнет вывод текста с позиции, где за­кончился вывод текущей строки  **file –** объект файла, стандартный поток или другой объект, похожий на файл, куда будет выводиться текст. По умолчанию используется объект sys.stdout стандартного потока вывода. В этом аргументе можно передать любой объект, поддерживающий метод файлов write(string);если передает­ся настоящий объект файла, он должен уже быть открыт для записи |
| **vars()** | vars()  {‘food’: ‘spam’, ‘age’: 40, **...**и еще множество других**...** } | **возвращает словарь, содержащий все переменные, существую­щие на момент ее вызова**  **ИДЕЯ: Выбор всех переменных в словаре для експорта параметров** |
| **Isinstance()** | Isinstance(x, list)  Isinstance(x, string)  Isinstance(x, tuple) | Проверяет является ли х списком  Строкой  Кортежем  словарем |
| **map()** | old\_list = ['1', '2', '3', '4', '5', '6', '7']  new\_list = list(**map(int, old\_list)**)   [1, 2, 3, 4, 5, 6, 7]  list (map (funk, [1, 2, 3], [2, 3, 4]))  # 1\*\*2, 2\*\*3, 3\*\*4  [1, 8, 81]  map ((lambda x: x + 3) , counters)) [4, 5, 6, 7] | Применить любую функцию к каждому элементу последовательности. В данном случае перевод с строки в целые. Принимает два аргумента: функцию и аргумент составного типа данных, например, список.  Поскольку **mар** применяет к каждому элементу вызов *функции,* а не произвольное *выражение,* она является менее универсальным инструментом и часто требует добавочных вспомогательных функций или lambda. |
| **zip()** | **L1 = [1,2,3,4]**  **L2 = [5,6,7,8]**  **list(zip(L1,L2))**  [(1, 5), (2, 6), (3, 7), (4, 8)]  **for (x, y) in zip(L1, L2):**  **print(x, y, ‘--’, x+y)**  1 5 -- 6  2 6 -- 8  3 7 -- 10  4 8 -- 12 | **Функция ЗИП** позволяет комбинировать последовательности создавая итератор с кортежами обоих элементов.  **Длина списка, возвращаемого функцией zip, равна длине кратчайшей из по­следовательностей, если аргументы имеют разную длину.**  S1 = ‘abc’  S2 = ‘xyz**123**’  list(zip(S1, S2))  [(‘a’, ‘x’), (‘b’, ‘y’), (‘c’, ‘z’)] |
| **enumerate()** | **S = ‘spam’**  **for (index, value) in enumerate(S):**  **print(value, ‘значение и индекс’, index)**  s значение и индекс 0  p значение и индекс 1  a значение и индекс 2  m значение и индекс 3  **E = enumerate(S)**  **next(E)**  (0, ‘s’)  **next(E)**  (1, ‘p’) | **Функция enumerate(S)** генерирует итератор из индекса и значения каждого элемента последовательности  Встроенная функция next() перемещает итератор на следующий элемент( \_\_next\_\_) |
| **filter()** | mixed = ['мак', 'просо', 'мак', 'мак', 'просо', 'мак', 'просо', 'просо', 'просо', 'мак']  zolushka = list(filter(lambda x: x == 'мак', mixed))  ['мак', 'мак', 'мак', 'мак', 'мак']  list (  filter (  (lambda x: x > 0) , range (-5, 5)))  [1, 2, 3, 4] | **Функция filter** предлагает элегантный вариант фильтрации элементов последовательности. Принимает в качестве аргументов функцию и последовательность, которую необходимо отфильтровать.  Обратите внимание, что функция, передаваемая в **filter** должна возвращать значение **True / False,** чтобы элементы корректно отфильтровались. |
|  |  |  |
|  |  |  |