ИТЕРАТОРЫ

|  |  |
| --- | --- |
| **I = iter(L)** | Создание итерируемого обекта |
| **I.\_\_next\_\_()**  или  **next(I)** | Переход к следующему элементу итератора |
| **L = [1, 2, 3]**  **I = iter(L)**  **I.\_\_next\_\_()**  1  **I.\_\_next\_\_()**  2 | *# Получить объект-итератор*  *# Вызвать \_\_next\_\_, чтобы перейти к следующему элементу*  При работе с файлами этот **iter(L)** начальный этап не нужен, потому что объект фай­ла имеет собственный итератор |
| **f = open(‘script1.py’)**  **f.\_\_next\_\_()**  ‘import sys\n’  **f.\_\_next\_\_()**  ‘print(sys.path)\n’  **f.\_\_next\_\_()**  ‘x = 2\n’  **f.\_\_next\_\_()**  **Traceback: Исключение когда строка пустая в конце файла** | *# Метод \_\_next\_\_ загружает одну строку*  *# и возбуждает исключение по достижении конца файла*  *Работает только в файлах без присваивания обекта итератора.* |
| **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’)] |
| **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\_\_) |
| **>>> list(open(‘script1.py’))**  [‘import sys\n’, ‘print(sys.path)\n’, ‘x = 2\n’, ‘print(2 \*\* 33)\n’]  **>>> tuple(open(‘script1.py’))**  (‘import sys\n’, ‘print(sys.path)\n’, ‘x = 2\n’, ‘print(2 \*\* 33)\n’)  **>>> ‘&&’.join(open(‘script1.py’))**  ‘import sys\n&&print(sys.path)\n&&x = 2\n&&print(2 \*\* 33)\n’  **>>> a, b, c, d = open(‘script1.py’)**  **>>> a, d**  (‘import sys\n’, ‘print(2 \*\* 33)\n’)  **>>> a, \*b = open(‘script1.py’) *# Расширенная форма в 3.0***  **>>> a, b**  (‘import sys\n’, [‘print(sys.path)\n’, ‘x = 2\n’, ‘print(2 \*\* 33)\n’]) |  |
|  |  |
| **[х \*\* 2 for х in range(4)]**  **(х \*\* *2* for х in range(4))** | ***# Списковое включение: строит список***  **[О, 1, 4, 9]**  ***# Генераторное выражение:* *создает итерируемый объект*** |
| **G = (х \*\* 2 for х in range (4) )**  **iter(G) is G**  **True**  **next(G)**  **О**  **ext(G)**  **1**  **next(G)**  **4**  **next(G)**  **9**  **next(G)** | ***# Вызов iter (G) необязателен: \_\_iter\_\_ возвращает сам объект***  ***# Объекты генераторов: автоматические методы*** |
| **list (map (abs, map (abs, map (abs, (-1,0,1)))))**  **list (abs (х) for х in (abs (x) for x in (abs (x) for x in (-1, 0, 1))))** | Списковые включения  Генераторы |
| **def funk():**  **for i in range(10):**  **yield  i**  **x=funk()**  **print(next(x))**  **print(next(x))**  **print(next(x))** | Функция yield возвращает генератор к которому можно получать доступ на каждом этапе итерации.  Вызов функции генератора выполняется через переменную. Так как прямой вызов функции возвращает всегда первое значение. А через переменную сохраняется состояние. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |