Поиск максимального значения в списке на Python

В этой статье мы научимся находить максимальное значение в списке на Python. Для всестороннего понимания вопроса мы рассмотрим использование некоторых встроенных функций, простые подходы, а также небольшие реализации известных алгоритмов.

Сначала давайте вкратце рассмотрим, что такое список в Python и как найти в нем максимальное значение или просто наибольшее число.

Список в Python

В Python есть встроенный тип данных под названием список (list). По своей сути он сильно напоминает массив. Но в отличие от последнего данные внутри списка могут быть любого типа (необязательно одного): он может содержать целые числа, строки или значения с плавающей точкой, или даже другие списки.

Хранимые в списке данные определяются как разделенные запятыми значения, заключенные в квадратные скобки. Списки можно определять, используя любое имя переменной, а затем присваивая ей различные значения в квадратных скобках. Он является упорядоченным, изменяемым и допускает дублирование значений. Например:

```
КОПИРОВАТЬ
list1 = ["Виктор", "Артем", "Роман"]
list2 = [16, 78, 32, 67]
list3 = ["яблоко", "манго", 16, "вишня", 3.4]
```

Далее мы рассмотрим возможные варианты кода на Python, реализующего поиск наибольшего элемента в списке, состоящем из сравниваемых элементов. В наших примерах будут использоваться следующие методы/функции:

1. Встроенная функция `max()`
2. Метод грубой силы (перебора)
3. Функция `reduce()`
4. Алгоритм Heap Queue (очередь с приоритетом)
5. Функция `sort()`
6. Функция `sorted()`
7. Метод хвостовой рекурсии

№1 Нахождение максимального значения с помощью функции max()

Это самый простой и понятный подход к поиску наибольшего элемента. Функция Python `max()` возвращает самый большой элемент итерабельного объекта. Ее также можно использовать для поиска максимального значения между двумя или более параметрами.

В приведенном ниже примере список передается функции max в качестве аргумента.

```
КОПИРОВАТЬ
list1 = [3, 2, 8, 5, 10, 6]
max_number = max(list1)
print("Наибольшее число:", max_number)
```

```
Наибольшее число: 10
```

КОПИРОВАТЬ

```
list1 = ["Виктор", "Артем", "Роман"]
max_string = max(list1, key=len)
print("Самая длинная строка:", max_string)
```

```
Самая длинная строка: Виктор
```

№2 Поиск максимального значения перебором

Это самая простая реализация, но она немного медленнее, чем функция `max()`, поскольку мы используем этот алгоритм в цикле.

В примере выше для поиска максимального значения нами была определена функция `large()`. Она принимает список в качестве единственного аргумента. Для сохранения найденного значения мы используем переменную `max_`, которой изначально присваивается первый элемент списка. В цикле for каждый элемент сравнивается с этой переменной. Если он больше `max_`, то мы сохраняем значение этого элемента в нашей переменной. После сравнения со всеми членами списка в `max_` гарантировано находится наибольший элемент.

КОПИРОВАТЬ

```
def large(arr):
    max_ = arr[0]
    for ele in arr:
        if ele > max_:
            max_ = ele
    return max_


list1 = [1,4,5,2,6]
result = large(list1)
print(result)  # вернется 6
```

№3 Нахождение максимального значения с помощью функции reduce()

В функциональных языках `reduce()` является важной и очень полезной функцией. В Python 3 функция `reduce()` перенесена в отдельный модуль стандартной библиотеки под названием functools. Это решение было принято, чтобы поощрить разработчиков использовать циклы, так как они более читабельны. Рассмотрим приведенный ниже пример использования `reduce()` двумя разными способами.

В этом варианте `reduce()` принимает два параметра. Первый — ключевое слово max, которое означает поиск максимального числа, а второй аргумент — итерабельный объект.

КОПИРОВАТЬ

```
from functools import reduce


list1 = [-1, 3, 7, 99, 0]
print(reduce(max, list1))  # вывод: 99
```

максимального значения.

```
from functools import reduce


list1 = [-1, 3, 7, 99, 0]
print(reduce(lambda x, y: x if x > y else y, list1))  # -> 99
```

№4 Поиск максимального значения с помощью приоритетной очереди

Heapq — очень полезный модуль для реализации минимальной очереди. Если быть более точным, он предоставляет реализацию алгоритма очереди с приоритетом на основе кучи, известного как heapq. Важным свойством такой кучи является то, что ее наименьший элемент всегда будет корневым элементом. В приведенном примере мы используем функцию `heapq.nlargest()` для нахождения максимального значения.

```
import heapq


list1 = [-1, 3, 7, 99, 0]
print(heapq.nlargest(1, list1))  # -> [99]
```

Приведенный выше пример импортирует модуль heapq и принимает на вход список. Функция принимает `n=1` в качестве первого аргумента, так как нам нужно найти одно максимальное значение, а вторым аргументом является наш список.

№5 Нахождение максимального значения с помощью функции sort()

Этот метод использует функцию `sort()` для поиска наибольшего элемента. Он принимает на вход список значений, затем сортирует его в порядке возрастания и выводит последний элемент списка. Последним элементом в списке является `list[-1]` .

```
list1 = [10, 20, 4, 45, 99]
list1.sort()
print("Наибольшее число:", list1[-1])
```

```
Наибольшее число: 99
```

№6 Нахождение максимального значения с помощью функции sorted()

Этот метод использует функцию `sorted()` для поиска наибольшего элемента. В качестве входных данных он принимает список значений. Затем функция `sorted()` сортирует список в порядке возрастания и выводит наибольшее число.

```
list1=[1,4,22,41,5,2]
sorted_list = sorted(list1)
```

№7 Поиск максимального значения с помощью хвостовой рекурсии

Этот метод не очень удобен, и иногда программисты считают его бесполезным. Данное решение использует рекурсию, и поэтому его довольно сложно быстро понять. Кроме того, такая программа очень медленная и требует много памяти. Это происходит потому, что в отличие от чистых функциональных языков, Python не оптимизирован для хвостовой рекурсии, что приводит к созданию множества стековых фреймов: по одному для каждого вызова функции.

КОПИРОВАТЬ

```
def find_max(arr, max_=None):
    if max_ is None:
        max_ = arr.pop()
    current = arr.pop()
    if current > max_:
        max_ = current
    if arr:
        return find_max(arr, max_)
    return max_


list1=[1,2,3,4,2]
result = find_max(list1)
print(result)  # -> 4
```

Заключение

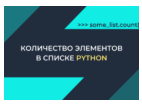
В этой статье мы научились находить максимальное значение из заданного списка с помощью нескольких встроенных функций, таких как `max()`, `sort()`, `reduce()`, `sorted()` и других алгоритмов. Мы написали свои код, чтобы попробовать метод перебора, хвостовой рекурсии и алгоритма приоритетной очереди.

Максим

Я создал этот блог в 2018 году, чтобы распространять полезные учебные материалы, документации и уроки на русском. На сайте опубликовано множество статей по основам python и библиотекам, уроков для начинающих и примеров написания программ.
Мои контакты: Почта

Статьи по теме

Нахождение делителей числа с помощью Python



Количество элементов в списке Python с условиями или критериями

Django + AJAX: как использовать AJAX в шаблонах Django

Как настроить Celery в Django

8 примеров использования value_counts из Pandas

СОДЕРЖАНИЕ

НОВОЕ В БЛОГЕ

Нахождение делителей числа с помощью Python
Лямбда-функции и анонимные функции в Python
Когда стоит использовать yield вместо return в Python
Как извлечь кубический корень в Python