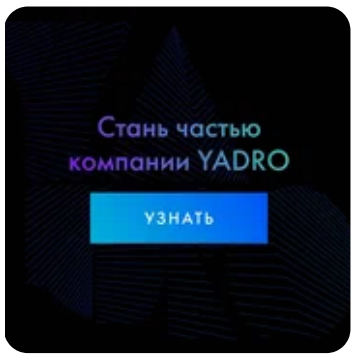
Сообщить об ошибке.

ХОЧУ ПОМОЧЬ ПРОЕКТУ

Подробное руководство по индексам и срезам в Python

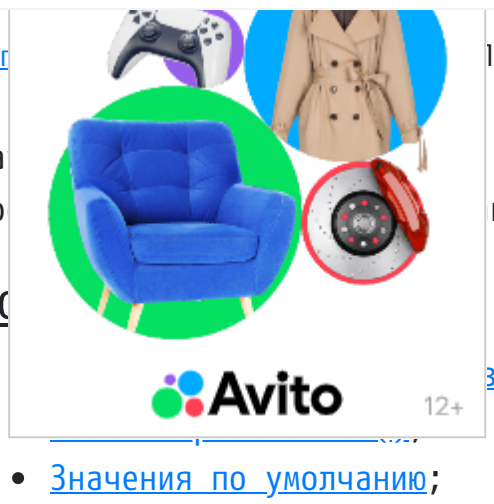


oneweekoffer.yadro.com РЕКЛАМА

Ищем ведущих программистов в команду YADRO.

Обучение и развитие • Открытое общение • Удаленно

Узнать больше



Cr Подробное руководство по индексам и срезам в Python

Ma как связан индекс элемента и срез последовательности, объясняется работа с объектом
ср ктике.

Co

- зы;
- Значения по умолчанию;
- Особый случай среза: копия последовательности;
- Примеры часто используемые срезов;
- Метод среза .indexes;
- Собственные классы с возможностью применения срезов;

> *Также смотрите:*
>
> - *Получение среза последовательности sequence[i:j];*
> - *Получение среза с заданным шагом sequence[i:j:k];*
> - *Замена части списка операцией среза;*
> - *Замена части списка срезом с заданным шагом;*
> - *Удаление части списка операцией среза;*
> - *Удаление части списка срезом с заданным шагом*

Введение в индексы и срезы.

В Python некоторые объекты, такие как строки или списки, можно нарезать. Например, можно получить первый элемент списка или строку с помощью индекса элемента:

```
>>> my_list = [1,2,3]
>>> my_list[0]
# 1

>>> my_string = "Python"
>>> my_string[0]
# P
```

Python использует квадратные скобки [и] для доступа к отдельным элементам объектов, которые можно разбить на части. Внутри этих квадратных скобок можно указать нечто большее, чем просто доступ к отдельным элементам.

Python поддерживает указание отрицательных индексов, их можно использовать следующим образом:

```
>>> my_list = list("Python")
>>> my_list[-1]
```

Выражение my_list[-1] представляет последний элемент списка, my_list[-2] представляет предпоследний элемент и так далее.

Что делать, если нужно извлечь более одного элемента последовательности? Например, необходимо извлечь все от начала до конца, за исключением самого последнего элемента:

```
>>> my_list = list("Python")
>>> my_list[0:-1]
```

Или, если нужен каждый *четный* элемент списка, то есть элемент 0, 2 и т. д.? Для этого нужно перейти от первого элемента к последнему, но пропустить каждый второй элемент. Например:

РЕКЛАМА

```
>>> my_list = list("Python")
>>> my_list[0:len(my_list):2]
# ['P', 't', 'o']
```

Объект среза slice().

За кулисами, индекс, который используется для доступа к отдельным элементам последовательности представляет собой объект среза slice() принимающий три аргумента: slice(start, stop, step).

Проверим это утверждение:

```
>>> my_list = list("Python")
>>> start = 0
>>> stop = len(my_list)
>>> step = 2
# объект среза
>>> slice_obj = slice(start, stop, step)
# сравниваем
>>> my_list[start:stop:step] == my_list[slice_obj]
# True
```

Примечание: буква P является первым элементом в списке (list("Python")), поэтому она индексируется как 0. Список имеет длину 6, и, следовательно, первый элемент также может быть проиндексирован как -6.

Если использовать начало start и конец stop объекта среза, то каждый элемент между этими числами покрывается срезом. Некоторые примеры:

```
>>> my_string = "Python"
>>> my_string[0:1]
# P
>>> my_string[0:5]
# Pytho
```

Это способ запомнить, что начальное значение является инклюзивным, а конечное - исключающим.

Значения по умолчанию.

В большинстве случаев приходится делить список:

- начиная с 0 (первый элемент последовательности);
- с захватом нужного последнего элемента;
- с определенным шагом.

Следовательно, значения по умолчанию можно опустить и использовать синтаксис ::

```
>>> my_list = list("Python")
>>> my_list[0:4] == my_list[:4]
# True

# используем шаг 2
>>> my_list[0:len(my_list):2] == my_list[::2]
# True
```

Технически, всякий раз, когда опускается число между двоеточиями, пропущенные числа будут иметь значение None.

И, в свою очередь, объект slice() заменит None на:

- 0 для начального значения;
- len(list) для конечного значения;
- 1 для значения шага.

Но, если значение шага отрицательное, то числа заменяются на:

- -1 для начального значения;
- -len(list) - 1 для конечного значения.

РЕКЛАМА

Например, 'Python'[::-1] технически совпадает с 'Python'[-1:-7:-1]

Особый случай среза: копия последовательности.

Существует специальный случай для среза, который можно использовать в качестве поверхностного копирования. Если использовать только значения по умолчанию, т. е. my_list[:], то это выражение создаст точно такие же элементы:

```
>>> my_list = list("Python")
>>> my_list_2 = my_list[:]
>>> my_list==my_list_2
# True
```

Элементы в списке действительно совпадают. Однако новый объект списка не является ссылкой на оригинал. Можно проверить это утверждение, используя встроенный идентификатор:

```
>>> id(my_list)
# 139967020891648
>>> id(my_list_2)
# 139967018223424
```

Обратите внимание, что каждая операция среза возвращает новый объект. Копия последовательности создается при использовании только [:].

Пример, иллюстрирующий разницу:

```
>>> a = list("Python")
# ссылка на оригинал списка
>>> b = a
>>> a[-1] = "N"
>>> a
# ['P', 'y', 't', 'h', 'o', 'N']
>>> b
# ['P', 'y', 't', 'h', 'o', 'N']

>>> a = list("Python")
# копия оригинала списка
>>> b = a[:]
>>> a[-1] = "N"
>>> a
# ['P', 'y', 't', 'h', 'o', 'N']
>>> b
# ['P', 'y', 't', 'h', 'o', 'n']
```

Примеры часто используемые срезов:

- [:] - копия последовательности;
- [::2] - четные элементы последовательности начиная с первого;
- [1::2] - нечетные элементы последовательности начиная со второго;
- [1:] - все элементы, кроме первого;
- [:-1] - все элементы, кроме последнего;
- [1:-1] - все элементы, кроме первого и последнего;
- [::-1] - все элементы в обратном порядке (реверс последовательности);
- [-2:0:-1] - все элементы, кроме первого и последнего, в обратном порядке;
- [-2:0:-2] - каждый второй элемент, кроме первого и последнего, в обратном порядке;

Метод среза .indices.

Каждый объект slice() в Python имеет метод slice.indices(). Этот метод возвращает кортеж (start, end, step), с помощью которой можно перестроить цикл, эквивалентный операции среза. Звучит сложно? Начнем разбираться с последовательности:

```
>>> sequence = list("Python")
```

Затем создадим объект slice(). Например возьмем каждый второй элемент, т.е. sequence[::2].

```
# эквивалентно `[::2]`
>>> my_slice = slice(None, None, 2)
```

РЕКЛАМА

Так как в качестве некоторых аргументов используется None, то объект slice() должен вычислять фактические значения индекса на основе длины последовательности. Следовательно, чтобы получить кортеж индексов, необходимо передать длину последовательности методу slice.indices(), например:

```
>>> indices = my_slice.indices(len(sequence))
>>> indices
# (0, 6, 2)
```

Теперь можно воссоздать цикл следующим образом:

```
sequence = list("Python")
start, stop, step = (0, 6, 2)
i = start
while i != stop:
    print(sequence[i])
    i = i+step
```

Этот цикл позволяет получить доступ к тем же элементам последовательности, что и сам срез.

Собственные классы с возможностью применения срезов.

Python не был бы Python, если бы невозможно было использовать объект среза в своих собственных классах. Более того, срезы не обязательно должны быть числовыми значениями. Например, можно создать адресную книгу, которую потом можно нарезать по алфавитным индексам.

```
import string

class AddressBook:
    def __init__(self):
        self.addresses = []

    def add_address(self, name, address):
        self.addresses.append((name, address))

    def get_addresses_by_first_letters(self, letters):
        letters = letters.upper()
        return [(name, address) for name, address in self.addresses
                if any(name.upper().startswith(letter) for letter in letters)]

    def __getitem__(self, key):
        if isinstance(key, str):
            return self.get_addresses_by_first_letters(key)
        if isinstance(key, slice):
            start, stop, step = key.start, key.stop, key.step
            letters = (string.ascii_uppercase[string.ascii_uppercase.index(start):string.ascii_uppercase.index(stop
            return self.get_addresses_by_first_letters(letters)

address_book = AddressBook()
address_book.add_address("Sherlock Holmes", "221B Baker St., London")
address_book.add_address("Wallace and Gromit", "62 West Wallaby Street, Wigan, Lancashire")
address_book.add_address("Peter Wimsey", "110a Piccadilly, London")
address_book.add_address("Al Bundy", "9764 Jeopardy Lane, Chicago, Illinois")
address_book.add_address("John Dolittle", "Oxenthorpe Road, Puddleby-on-the-Marsh, Slopshire, England")
address_book.add_address("Spongebob", "124 Conch Street, Bikini Bottom, Pacific Ocean")
address_book.add_address("Hercule Poirot", "Apt. 56B, Whitehaven Mansions, Sandhurst Square, London W1")
address_book.add_address("Bart Simpson", "742 Evergreen Terrace, Springfield, USA")


print(string.ascii_uppercase)
```

```
print(string.ascii_uppercase.index("A"))
print(string.ascii_uppercase.index("Z"))

print(address_book["A"])
print(address_book["B"])
print(address_book["S"])
print(address_book["A":"H"])
```

РЕКЛАМА

Разбираемся в созданном классе AddressBook.

- Метод get_addresses_by_first_letters():

  Этот метод фильтрует все адреса, принадлежащие имени, которые начинаются с любой буквы в аргументе letters. Во-первых, эта функция нечувствительна к регистру, так как преобразует буквы в верхний регистр. Затем используется генератор списка поверх внутреннего списка адресов. Условие внутри генератора списка проверяет, соответствует ли какая-либо из предоставленных букв первой букве, соответствующее значению имени.

- Метод __getitem__:

  Чтобы сделать объекты адресной книги доступными для использования среза, необходимо переопределить магический метод Python __getitem__.

  Сначала проверяется, является ли ключ строкой. Это будет иметь место, если получаем доступ к объекту с помощью одной буквы в квадратных скобках, например так: address_book['A']. Для этого тривиального случая можно просто вернуть любые адреса, имя которых начинается с данной буквы.

  Интересная часть, когда ключ является объектом среза. Например, этому условию будет соответствовать обращение типа address_book['A':'H']. Во-первых, идентифицируются все буквы в алфавитном порядке между буквами A и H. Модуль string перечисляет все латинские буквы в string.ascii_uppercase. Далее используется срез для извлечения букв между заданными буквами. Обратите внимание на +1 во втором параметре среза. Таким образом, гарантируется, что последняя буква является включающей, а не исключающей.

  После того, как определили все буквы в последовательности, используется метод get_addresses_by_first_letters(), о котором говорилось выше.

DOCS-Python.ru™, 2023 г. (Внимание! При копировании материала ссылка на источник обязательна) @docs_python_ru