# Методические указания

## Урок 5.1 Строки и их методы

**Задачи урока:**

- Повторить строковый тип данных
- Изучить индексацию строк
- Изучить срезы строк
- f-string

### 0. Подготовка к уроку

До начала урока преподавателю необходимо:

1) Просмотреть, как ученики справились с домашним заданием
2) Прочитать методичку

### 1. Строковый тип данных

**Учитель:** Любое число можно рассматривать как строку, последовательность символов. Возможность взаимодействовать с символами строки упрощает работу с цифрами чисел.

**Учитель:** Строковый тип данных, в Python представленный типом str, один из наиболее часто используемых в программировании. В числе инструментов работы с ним — индексация и срезы.
Давайте рассмотрим основные возможности при работе со строками

1.Перебор строки

```
word = 'hello'
for char in word:
    print(char)
```

2. Конкатенация (объединение) строк

```
word = 'hello'
print(word + ' hello')
```

3. Нахождение длины строки

```
word = 'hello'
print(len(word))
```

4. Проверка вхождения в строку

```
word = 'pencil'
if 'pen' in word:
    print('Yes')
```

5. Умножение строки на число

```
print('hello' * 3)
```

## 2. Индексация в строках

**Учитель:** Мы с вами знаем, что строки состоят из символов. Очень часто нам надо обратиться к конкретному символу в строке. Для этого в Python используются квадратные скобки [], в которых указывается индекс (номер) нужного символа в строке. Такая операция обращения по определенному индексу называется **индексацией**.
Рассмотрим строку:
`s = 'Python'`
Несложно предположить, что нумерация символов начинается с 0.

| Выражение | Результат | Пояснение |
|---|---|---|
| s[0] | P | первый символ строки |
| s[1] | y | второй символ строки |
| s[2] | t | третий символ строки |
| s[3] | h | четвертый символ строки |
| s[4] | o | пятый символ строки |
| s[5] | n | шестой символ строки |

**Учитель:** Такой подход в нумерации символов в строке является стандартным практически во всех языках программирования. Скажем в C#, Java, C++ и т.д. нумерация также

начинается с нуля. Такой подход удобен, так как мы легко по индексу (порядковому номеру) символа можем определить сам символ в строке. Разработчики Python пошли еще дальше и добавили возможность обращения по отрицательным индексам. Если первый символ строки имеет индекс 0, то последнему символу присваивается индекс -1. Предпоследний символ имеет индекс -2, и так далее до начала строки.

| Выражение | Результат | Пояснение |
|---|---|---|
| `s[-6]` | `P` | `первый символ строки` |
| `s[-5]` | `y` | `второй символ строки` |
| `s[-4]` | `t` | `третий символ строки` |
| `s[-3]` | `h` | `четвертый символ строки` |
| `s[-2]` | `o` | `пятый символ строки` |
| `s[-1]` | `n` | `шестой символ строки` |

**Учитель:** Следует сказать, что отрицательные индексы это особенность именно языка Python. С их помощью можно одинаково легко обращаться к последнему, предпоследнему и т.д. символу строки не зная даже ее длины.

**Учитель:** Очень часто программисты могут совершать ошибки выхода за пределы строки. В таком случае в Python мы получаем ошибку: **IndexError: string index out of range**.

**Учитель:** Давайте порешаем устные задачи.

**Задача 1.** Что покажет приведенный ниже фрагмент кода?

```
s = 'abcdefg'
print(s[0] + s[2] + s[4] + s[6])
```

**Решение.** Результатом выполнения такой программы будет конкатенация строк `'a'`, `'c'`, `'e'`, `'g'`. То есть строка 'aceg'.

**Ответ:** aceg

**Задача 2.** Что покажет приведенный ниже фрагмент кода?

```
s = 'abcdefg'
print(s[0]*3 + s[-1]*3 + s[3]*2 + s[3]*2)
```

**Решение.** Результатом выполнения такой программы будет конкатенация строк `'aaa'`, `'ggg'`, `'dd'`, `'dd'`. То есть строка 'aaagggdddd'.

**Ответ:** aaagggdddd

**Учитель:** Имея механизм обращения к отдельным символам строки по индексу, а также циклы мы можем просканировать строку целиком, обрабатывая каждый символ. Другими словами мы можем совершать итерацию по строке. Существует два основных способов итерирования строк. Рассмотрим первый способ: итерация по индексам:

```
s = 'abcdef'
for i in range(len(s)):
    print(s[i])
```

Мы передаем в функцию range, длину строки, вызывая функцию len. Значит переменная i последовательно принимает все значения из диапазона 0, 1, 2, …, len(s) - 1. В теле цикла мы обращаемся к i-тому индексу с помощью индексатора. Таким образом мы распечатаем все символы строки по отдельности, каждый на отдельной строке. **Такой способ итерации строки удобен, когда нам нужен не только сам элемент s[i], но и его индекс i.**

**Учитель:** В Python есть еще один способ итерации. Рассмотрим следующий код:

```
s = 'abcdef'
for c in s:
    print(c)
```

Этот цикл пройдет по строке s, придавая переменной цикла с значение каждого символа (!) в отличии от предыдущего цикла, в котором переменная цикла «бегала» по индексам строки. Такой способ итерации более короткий, однако его применять мы можем тогда, когда нам не нужны индексы, а нужны только сами элементы.

**Учитель:** Первый подход, когда мы итерировали по индексам является более функциональным, так как при таком подходе у нас есть доступ и к индексам i и к самим элементам строки s[i]. Однако такой код чуть более громоздкий. Ученикам следует обратить внимание на именование переменных цикла:

- в первом цикле мы используем имя i, что соответствует стандартной идеологии наименования переменных цикла;
- во втором цикле, мы назвали переменную буквой c – первая буква слова char (символ).

**Ученикам предлагается решить устную задачу.**

**Задача.** Что покажет приведенный ниже фрагмент кода?

```
s = '012345678910111213141516l7'
for i in range(0, len(s), 5):
    print(s[i], end='')
```

**Решение.** Такой код покажет каждый пятый символ строки s, так как шаг цикла равен 5. То есть будет выведена строка: '051217'

**Ответ:** 051217

## 3. Срезы строк

**Учитель:** Иногда нужно работать с целыми частями строки, в таком случае мы используем срезы (slices). С помощью среза мы можем получить несколько символов исходной строки, создав диапазон индексов разделенных двоеточием s[x:y].

Рассмотрим строку `s = 'abcdefghij'`.

| **Индексы** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Строка** | a | b | c | d | e | f | g | h | i | j |
| **Индексы** | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

Следующий код:

```
s = 'abcdefghij'
print(s[2:5])
print(s[0:6])
print(s[2:7])
```

Выводит:

```
cde
abcdef
cdefg
```

**Учитель:** В срезе s[x:y] первое число – это то место, где начинается срез (включительно), а второе – это место, где заканчивается срез (невключительно).

**Учитель:** Очень часть нам нужно получить срез до конца строки или срез от начала строки. В таком случаем мы можем опустить один из параметров x или y.
Если опустить второй параметр в срезе s[x:], то срез берется до конца строки.

Если опустить первый параметр s[:y], то можно взять срез от начала строки.

**Учитель:** Мы можем использовать отрицательные индексы для создания срезов. При использовании отрицательных индексов первый параметр среза должен быть меньше второго, либо должен быть пропущен. Следующий код

```
s = 'abcdefghij'
print(s[-9:-4])
print(s[-3:])
print(s[:-3])
```

выводит

```
bcdef
hij
abcdefg
```

Следует сказать, что отрицательные индексы в срезах на практике используются не так часто.

**Учитель:** Следует сказать, что возможно вы заметили схожесть срезов и функции range, которая генерирует последовательность целых чисел. При использовании функции range мы могли передать третий параметр, отвечающий за шаг генерации чисел. Аналогично мы можем и в срезах указывать шаг среза. Например, следующий код:

```
s = 'abcdefghij'
print(s[1:7:2])
```

выведет строку `bdf`.

**Учитель:** Ну и наконец относительно срезов следует знать про отрицательный шаг в срезе. Отрицательный шаг среза позволяет получить символы строки в обратном порядке. Рассмотрим следующий код:

```
s = 'abcdefghij'
print(s[::-1])
print(s[::-2])
```

Такой код выведет:

```
jihgfedcba (строка в обратном порядке)
jhfdb (каждый второй символ начиная с конца)
```

Хотелось бы уделить немного времени простой, но важной теме - f строкам. За время написания кода вы уже наверное устали связывать вывод строки и переменных через символ +. Этот метод(конкатенация) более длинный и неудобный для чтения. Существуют несколько методов форматирования текста, но мы рассмотрим f строку. Реализуется она очень просто. Всего лишь достаточно указать букву f перед кавычками, а все переменные брать в фигурные скобки.

```
age = 15
name = ‘Иван’
print(f’Привет меня зовут {name}, мне {age} лет’)
```

Как видим f строка более просто читается и в ней достаточно сложно запутаться.

## 4. Решение задач

Задача 1

Написать программу, проверяющую является ли строка палиндромом. Палиндром - строка, которая читается одинаково, как слева направо, так и наоборот. Например слово шалаш

Решение

```
word = ‘шалаш’
if word == word[::-1]:
    print(‘Yes’)
else:
    print(‘No’)
```

## Дополнительно

Если на уроке остается время, то ученикам можно предложить начать прорешивать домашнее задание.

## Домашняя работа

Задача 1