

## **ПМЗ Разработка модулей ПО.**

**РО 3.1 Понимать и применять принципы объектно-ориентированного и асинхронного программирования.**

# **Тема 5. Строки. Методы строк.**

**Цель занятия:**

**Сформировать представление о строках как о базовом типе данных в Python.**

**Научиться использовать основные методы строк для обработки текстовой информации.**

# **Учебные вопросы:**

- 1. Понятие строки в Python**
- 2. Основные методы строк в Python**
- 3. f-строки**
- 4. Экранирование символов и специальные символы в строках**

# 1. Понятие строки в Python

Строка (string, str) — это последовательность символов, используемая для хранения текста в Python.

Символами могут быть буквы, цифры, знаки препинания, пробелы и специальные символы.

Строки — один из базовых типов данных в Python.

Строка неизменяема (immutable): нельзя изменить отдельный символ строки, можно только создать новую строку на основе существующей.

## Литералы строк:

```
# Одинарные кавычки '...'
```

```
text = 'Hello'
```

```
# Двойные кавычки "..."
```

```
text = "Python"
```

```
# Тройные кавычки '''...''' или """..."""
```

```
# Позволяют писать многострочные строки.
```

```
text = """Hello
```

```
,
```

```
World"""
```

## Индексация и срезы строк

Каждый символ строки имеет индекс.

Индексы начинаются с 0 (первый символ) и идут вправо.

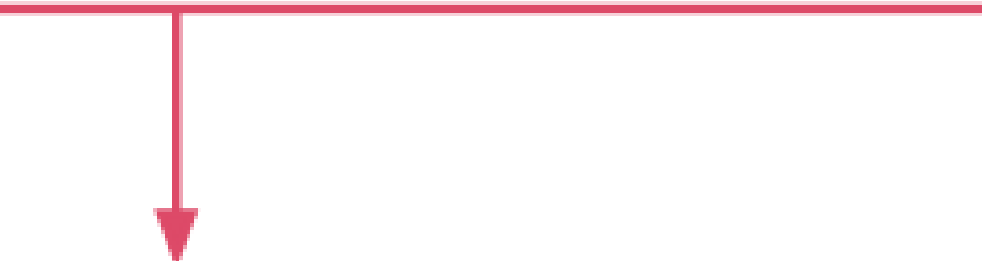
Можно использовать отрицательные индексы, чтобы идти с конца (-1 — последний символ):

```
s = "Python"
print(s[0])      # P
print(s[5])      # n
print(s[-1])     # n
print(s[-2])     # o
```

# Индексация и срезы строк

Доступ к элементам объекта по их порядковому номеру в нем.

Индексация элементов **начинается с нуля.**



0	1	2	3	4	5
И	Н	Д	Е	К	С
-6	-5	-4	-3	-2	-1

# Поиск символов

Получить значение элемента по индексу можно при помощи `[ ]`.

`my_string[0]`

или

`my_string[-6]`

0	1	2	3	4	5
И	Н	Д	Е	К	С
-6	-5	-4	-3	-2	-1

## Срезы (slicing)

Срез позволяет получить часть строки: `s[start:stop:step]`

- `start` — начальный индекс (включительно)
- `stop` — конечный индекс (не включительно)
- `step` — шаг (по умолчанию 1)

Примеры:

```
s = "Python"
print(s[0:4])      # Pyth
print(s[2:])       # thon
print(s[:4])       # Pyth
print(s[::2])      # Pto
print(s[::-1])     # nohtyP (обратная строка)
```

Можно извлечь из строки несколько элементов при помощи “срезов” (slicing). Для указания интервала среза используется `:`.

Синтаксис: `string[start:stop]`, где

`start` – индекс первого элемента в списке,

`stop` – индекс списка, перед которым срез должен закончиться (т.е. сам элемент с индексом `stop` не будет входить в выборку).

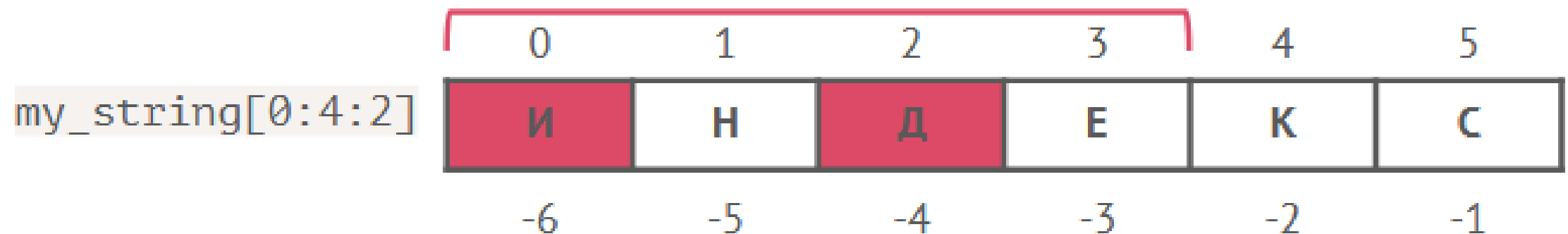
`my_string[1:3]`

0	1	2	3	4	5
И	Н	Д	Е	К	С
-6	-5	-4	-3	-2	-1

Срез с шагом. Шаг указывает, на сколько символов нужно подвинуться после взятия первого символа.

Синтаксис: `string[start:stop:step]`, где

`step` – шаг прироста выбираемых индексов.



`string[start:stop:step]`

При этом любой из параметров может быть опущен. Тогда вместо соответствующего параметра будет выбрано значение по умолчанию:

- `start` по-умолчанию означает «от начала списка»,
- `stop` по-умолчанию означает «до конца списка» (включительно),
- `step` по-умолчанию означает «брать каждый элемент».

`my_string[3:]`

0	1	2	3	4	5
И	Н	Д	Е	К	С
-6	-5	-4	-3	-2	-1

`my_string[:3]`

0	1	2	3	4	5
И	Н	Д	Е	К	С
-6	-5	-4	-3	-2	-1

## Неизменяемость строк (immutable)

После создания строки нельзя изменить отдельный СИМВОЛ:

```
s = "Hello"  
s[0] = "h"    # Ошибка!
```

Чтобы изменить строку, создают новую строку:

```
s = "Hello"  
s_new = "h" + s[1:]  
print(s_new)  # hello
```

## 2. Основные методы строк в Python

Метод — это функция, которая «принадлежит» определённому объекту и используется для выполнения действий с этим объектом.

Каждый объект в Python (строка, список, словарь и др.) имеет свои методы.

Методы вызываются через точку после имени объекта:  
**объект.метод()**

Строки (str) в Python имеют множество встроенных методов, которые упрощают их обработку.

Методы позволяют изменять регистр, проверять содержимое, искать и заменять текст, удалять пробелы и др.

## Методы изменения регистра

Метод	Описание	Пример
<code>upper()</code>	Преобразует все символы в верхний регистр	<code>"hello".upper() → "HELLO"</code>
<code>lower()</code>	Преобразует все символы в нижний регистр	<code>"HELLO".lower() → "hello"</code>
<code>title()</code>	Преобразует первую букву каждого слова в верхний регистр	<code>"hello world".title() → "Hello World"</code>
<code>capitalize()</code>	Делает первую букву строки заглавной, остальные — строчными	<code>"python".capitalize() → "Python"</code>

## Булевы методы проверки содержимого

Метод	Описание	Пример
<code>isalpha()</code>	Все символы — буквы	<code>"abc".isalpha() → True</code>
<code>isdigit()</code>	Все символы — цифры	<code>"123".isdigit() → True</code>
<code>isalnum()</code>	Все символы — буквы или цифры	<code>"abc123".isalnum() → True</code>
<code>isspace()</code>	Все символы — пробельные	<code>" ".isspace() → True</code>
<code>istitle()</code>	Строка оформлена в виде заголовка	<code>"Hello World".istitle() → True</code>

## Методы поиска и анализа

Метод	Описание	Пример
<code>find(sub)</code>	Возвращает индекс первого вхождения <code>sub</code> , если нет — -1	<code>"Python".find("t") → 2</code>
<code>rfind(sub)</code>	Поиск с конца строки	<code>"Python".rfind("o") → 4</code>
<code>index(sub)</code>	Как <code>find</code> , но вызывает ошибку, если <code>sub</code> не найден	<code>"Python".index("o") → 4</code>
<code>count(sub)</code>	Подсчитывает количество вхождений подстроки <code>sub</code>	<code>"hello".count("l") → 2</code>

# Методы замены и удаления

Метод	Описание	Пример
<code>replace(old, new)</code>	Заменяет все вхождения <code>old</code> на <code>new</code>	<code>"hello".replace("l", "x") → "hexxo"</code>
<code>strip()</code>	Удаляет пробелы или указанные символы с начала и конца	<code>" hello ".strip() → "hello"</code>
<code>lstrip()</code>	Удаляет символы слева	<code>" hello".lstrip() → "hello"</code>
<code>rstrip()</code>	Удаляет символы справа	<code>"hello ".rstrip() → "hello"</code>

# Разделение и объединение строк

Метод	Описание	Пример
<code>split(sep)</code>	Разбивает строку на список по разделителю <code>sep</code>	<code>"a,b,c".split(",") → ['a','b','c']</code>
<code>rsplit(sep)</code>	Разбивает строку справа	<code>"a,b,c".rsplit(",", 1) → ['a,b','c']</code>
<code>join(iterable)</code>	Объединяет элементы последовательности в строку с разделителем	<code>",".join(["a","b","c"]) → "a,b,c"</code>
<code>rstrip()</code>	Удаляет символы справа	<code>"hello ".rstrip() → "hello"</code>

## Встроенная функция `len()`

Возвращает длину строки (количество символов).

```
s = "Python"
print(len(s))    # 6
```

## Оператор `in`

Проверяет, содержится ли подстрока в строке.

Возвращает `True` или `False`.

```
s = "Hello, world!"
print("Hello" in s)    # True
print("Python" in s)   # False
```

# 3. f-строки (форматированные строки)

f-строка — это способ встраивать значения переменных прямо в текст строки.

Синтаксис: перед строкой ставится буква f или F, а переменные заключаются в {}.

```
name = "Alice"  
age = 25  
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Alice and I am 25 years old.
```

## Использование выражений внутри f-строк

Внутри `{}` можно использовать арифметические операции, вызовы функций и методы объектов:

```
a = 5
b = 3
print(f"{a} + {b} = {a + b}")    # 5 + 3 = 8

name = "alice"
print(f"Hello, {name.upper()}!")  # Hello, ALICE!
```

## Форматирование чисел

Количество знаков после запятой для вещественных чисел:

```
pi = 3.1415926
print(f"Pi ≈ {pi:.2f}")    # Pi ≈ 3.14
print(f"Pi ≈ {pi:.4f}")    # Pi ≈ 3.1416
```

Что означает **{pi:.2f}**?

- **:** — начало спецификации формата
- **.2** — количество знаков после запятой
- **f** — формат числа как **float** (вещественное)

## Выравнивание текста:

```
name = "Bob"
print(f"{name:<10}!") # Выравнивание по левому краю
print(f"{name:>10}!") # Выравнивание по правому краю
print(f"{name:^10}!") # Выравнивание по центру
```

Bob !

Bob!

Bob !

Что означает :10 внутри фигурных скобок?

Запись **:10** задаёт ширину поля — то есть Python резервирует 10 символов под вывод строки.

Имя "Bob" занимает 3 символа, значит ещё 7 позиций — пустые.

**<10** → выровнять строку по левому краю в поле из 10 СИМВОЛОВ.

Результат:

Bob      !

Этот механизм используется для:

- красивых таблиц,
- отчётов,
- колонок данных,
- консольных интерфейсов.

## 4. Экранирование символов и специальные символы в строках

Экранирование — это способ указать Python, что символ должен интерпретироваться не буквально, а как часть специальной последовательности.

Экранирование выполняется с помощью обратного слэша \.

Пример:

```
print("He said: \"Hello\"")
```

```
He said: "Hello"
```

Здесь \" — экранированная кавычка.

Экранирование используется, когда нужно:

- вставить в строку кавычку того же типа, что ограничивает строку,
- использовать служебные символы (перевод строки, табуляция и др.),
- записывать непечатаемые или управляющие символы,
- включать в строку сам символ \.

# Escape-последовательности

Последовательность	Описание
\\	Обратный слеш. Выводит один знак обратного слеша
\'	Апостроф, или одиночная кавычка. Выводит один апостроф
\"	Двойные кавычки. Выводит одну такую кавычку
\n	Пустая строка. Перемещает курсор в начало следующей строки
\t	Горизонтальный отступ – символ табуляции. Перемещает курсор вправо на один отступ

Примеры:

```
print("Line 1\nLine 2")  
print("Column1\tColumn2")  
print("Slash: \\")
```

Line 1

Line 2

Column1 Column2

Slash: \

# Экранирование кавычек в строках

Примеры использования:

```
print('It\'s OK')  
print("He said: \"Hello\"")
```

Если используются разные типы кавычек, экранировать не обязательно:

```
print("It's OK")           # можно без экранирования  
print('He said: "Hi"')    # тоже нормально
```

## Сырые строки (raw strings)

Если перед строкой поставить **r**, экранирование не работает — все \ считаются обычными символами.

```
print(r"C:\new_folder\test")
```

```
C:\new_folder\test
```

Полезно для:

- регулярных выражений,
- Windows-путей,
- описания шаблонов.

Важно! Сырые строки не могут заканчиваться одинарным \:

## Итог:

- Экранирование позволяет корректно отображать специальные символы в строках.
- Основной инструмент — обратный слэш \.
- Сырые строки (r"") отключают экранирование.
- Escape-последовательности используются для управления форматированием текста, записи путей, обработки Unicode и других задач.

# Контрольные вопросы:

- Что такое строка в Python? Почему строки считаются неизменяемыми?
- Чем отличаются одинарные, двойные и тройные кавычки?
- Как работают индексы и срезы строк?
- Какая разница между `find()` и `index()`?
- Для чего используются методы `strip()`, `replace()` и `split()`?
- Как объединить список строк в одну строку?
- Что такое f-строки и какие преимущества они дают?
- Что означает "строка — итерируемый объект"?
- Приведите примеры булевых методов строк и объясните их назначение.
- Что такое экранирование символов? Приведите примеры.

# Домашнее задание:

<https://ru.hexlet.io/courses/js-asynchronous-programming>

# **Материалы лекций:**

<https://github.com/ShViktor72/Education>

# **Обратная связь:**

[colledge20education23@gmail.com](mailto:colledge20education23@gmail.com)