ISSN 2221-1993 • **Informatics in School** • 2025 • Volume 24 No 3

DOI: 10.32517/2221-1993-2025-24-3-84-90

Т. С. Гончарова

Донецкий Республиканский Дворец детского и юношеского творчества, г. Донецк, Донецкая Народная Республика, Россия

СТРОКИ В PYTHON: ОПРЕДЕЛЕНИЕ И ПРЕДСТАВЛЕНИЕ*

Аннотация

Статья посвящена строкам в Python — одному из основных типов данных, используемых для хранения и обработки текстовых значений. Рассматриваются способы определения строк, их создание с помощью одинарных или двойных кавычек, а также примеры однострочных и многострочных строк.

Для лучшего понимания внутреннего представления строк в Python рассмотрено сравнение строк — базовая операция, которая используется для определения порядка, равенства или различия между двумя строками. Рассмотрены основные операторы сравнения строк в Python, такие как «= =» (равно), «!=» (не равно), «<» (меньше), «>» (больше), «<=» (меньше или равно) и «>=» (больше или равно). Обсуждаются особенности работы со строками и некоторые нюансы, которые следует учитывать при сравнении строк.

Статья будет полезна при изучении типа данных «строка» всем, кто стремится улучшить свое понимание строковых данных в Python.

Ключевые слова: строки в Python, описание однострочных и многострочных строк, экранирующий символ, сравнение строк.

Информация об авторе

Гончарова Татьяна Семеновна, педагог дополнительного образования, Донецкий Республиканский Дворец детского и юношеского творчества, г. Донецк, Донецкая Народная Республика, Россия; *e-mail:* tavisevik@yandex.ru

1. Введение

В современном мире информационные технологии развиваются с невероятной скоростью, и знания в этой области становятся все более ценными. Преподавание информатики требует особого подхода, который не только позволит передать учащимся необходимые навыки, но и пробудит в них интерес к дальнейшему изучению предмета.

Важно понимать, что информатика — это не просто набор фактов и формул, а целая вселенная возможностей для творчества, исследований и инноваций [2]. Углубленное изучение материала позволяет учащимся не только лучше понять основы информатики, но и научиться применять их на практике, решать сложные задачи и находить нестандартные решения.

2. Примеры работы со строками в Python

2.1. Определение

Строки — это один из основных типов данных в Python, который используется для ввода, хранения и обработки текстовых данных. Строки представляют собой последовательности символов, которые могут

* Материалы к статье можно скачать на сайте ИНФО: http://infojournal.ru/journals/school/school_03-2025/

T. S. Goncharova

Donetsk Republican Palace of Children's and Youthful Creativity, Donetsk, The Donetsk People's Republic, Russia

STRINGS IN PYTHON: DEFINITION AND REPRESENTATION

Abstract

The article is about strings in Python, one of the main data types used to store and process text values. It covers ways to define strings, create them using single or double quotes, and examples of single-line and multi-line strings.

To better understand the internal representation of strings in Python, we consider string comparison, a basic operation used to determine the order, equality, or difference between two strings. We consider the main string comparison operators in Python, such as "= =" (equal), "!=" (not equal), "<" (less than), ">" (greater than), "<=" (less than or equal), and ">=" (greater than or equal). We discuss the specifics of working with strings and some nuances that should be taken into account when comparing strings.

The article will be useful in studying the "string" data type for anyone who wants to improve their understanding of string data in Python.

Keywords: strings in Python, description of single- and multi-line strings, escape character, comparing strings.

Information about the author

Tatyana S. Goncharova, a teacher of additional education, Donetsk Republican Palace of Children's and Youthful Creativity, Donetsk, The Donetsk People's Republic, Russia; *e-mail:* tavisevik@yandex.ru

ISSN 2221-1993 • **Информатика в школе** • 2025 • Том 24 № 3

быть созданы с помощью одинарных или двойных кавычек [1, 3, 4, 9]. Внутри этих кавычек можно указать любой текст, который будет интерпретирован как строка [11, 15].

Например:

```
string1 = 'Роман "Война и мир" Льва Толстого…'
string2 = "It's my book."
```

Эти две строки представляют собой текстовые однострочные данные.

2.2. Представление

Удобство использования разных кавычек хорошо видно на примерах, когда внутри строк нужно написать текст с использованием кавычек другого типа.

Использование одинаковых кавычек внутри и на краях строки допустимо, если перед внутренними кавычками поставить символ «обратный слеш» (\), экранирующий (изменяющий) действие следующего за ним символа [8, 9, 16, 17].

То есть эти представленные выше две строки можно было бы записать так:

```
string3 = "Роман \"Война и мир\" Льва
 Толстого…"
string4 = 'It\'s my book.'
```

Пара строк *string*1 и *string*2 и пара строк *string*3 и *string*4 при выводе дадут одинаковый результат, и длина строк также будет одинакова. Длину строки можно определить с помощью функции *len*(*string*) (рис. 1).

Для создания однострочной строки достаточно указать текст внутри кавычек:

```
single_line_string = 'Однострочная строка.'
```

Можно использовать тройные кавычки (''' или """) в начале и в конце строки, чтобы создать многострочную строку:

```
multi_line_string = '''Многострочная
 строка, которая может занимать несколько
 строк.'''
```

Этот способ позволяет создавать строки, которые занимают несколько строк кода. Это особенно полезно при работе с длинными текстами или при форматировании кода для лучшей читаемости.

Начинать и заканчивать строку необходимо кавычками одного типа.

Согласно правилам красивого кода PEP8 [20]:

- в программе для оформления строк необходимо использовать один вид кавычек;
- для документации кода используются тройные кавычки;
- при описании подпрограмм многострочная строка применяется как строка документирования сразу после заголовка функции.

Все это помогает другим программистам понимать код и использовать его в своих проектах.

2.3. Пример многострочной строки. Экранирующий символ

Для закрепления информации о многострочных строках создадим строку с простым рисунком внутри:

```
house = '''
      /\
     /  \
    /    \
   /      \
  /        \
  ----------
  |        |
  |  ---   |
  | |   | ||
  |  ---  ||
  |       ||
 ------------
 ------------
'''
print(house)
```

После запуска программы видим, что у домика стены стоят, а крыша «поехала» (рис. 2).

Из рисунка 2 видно, что символы «обратный слеш» (\), которыми была нарисована крыша, не вывелись. Так получилось, потому что в Python символ «\» по умолчанию экранирует (закрывает, изменяет действие следующего непосредственно за слешем

```
>>> string1 = 'Роман "Война и мир" Льва Толстого…'
>>> string3 = "Роман \"Война и мир\" Льва Толстого…"
>>> string1
'Роман "Война и мир" Льва Толстого…'
>>> string3
'Роман "Война и мир" Льва Толстого…'
>>> len(string1)
34
>>> len(string3)
34
>>> string2 = "It's my book."
>>> string4 = 'It\'s my book.'
>>> string2
"It's my book."
>>> string4
"It's my book."
>>> len(string2)
13
>>> len(string4)
13
```

Рис. 1. Пример создания и вывода строк в консоль

ISSN 2221-1993 • **Informatics in School** • 2025 • Volume 24 No 3

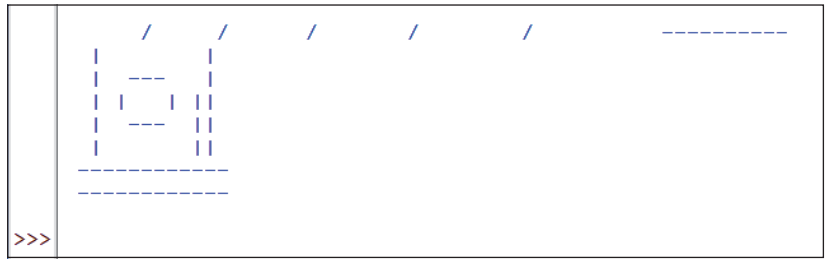

Рис. 2. Пример вывода многострочной строки «Домик» с ошибкой

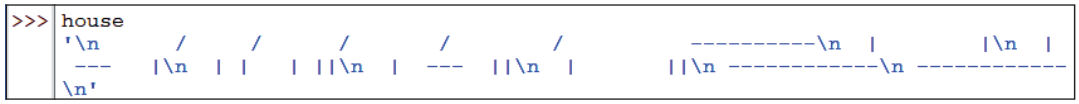

Рис. 3. Внутреннее представление многострочной строки «Домик» с ошибкой

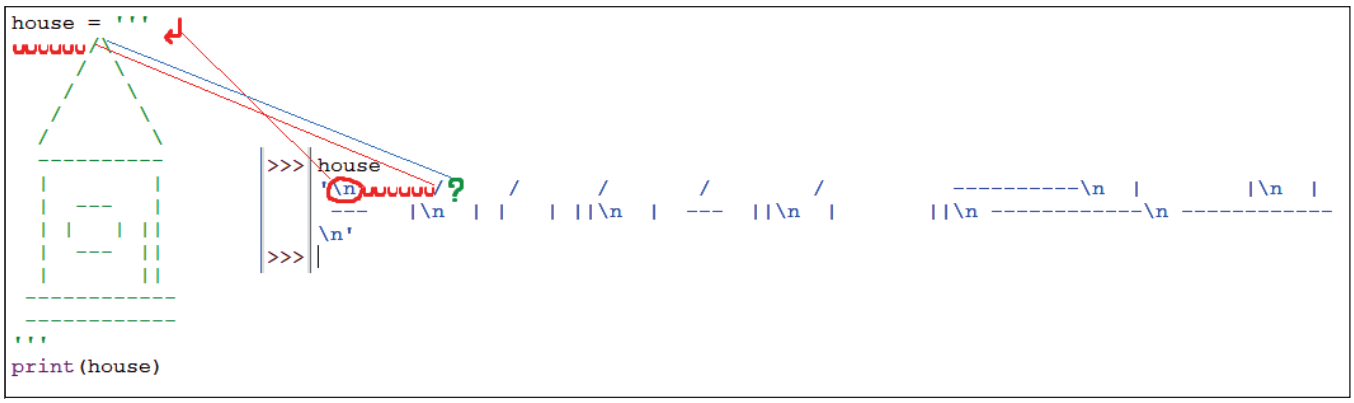

Рис. 4. Анализ внутреннего представления многострочной строки «Домик»

символа [1, 5, 9]) следующий символ. После каждого символа «\» была нажата клавиша Enter, переводящая курсор на следующую строку.

Посмотрим на внутреннее представление строки (рис. 3).

Сочетание «\n» — это перевод курсора на следующую строку (рис. 4).

Из рисунка 4 видно, что первый перевод курсора в строке отображен, дальнейшие шесть пробелов и прямой слеш присутствуют в сроке *house*. Но обратный слеш экранировал стоящий за ним перевод курсора, поэтому в строке отсутствуют обратный слеш и перевод курсора. Это же повторилось в каждой строке крыши домика. Поэтому и крыша «поехала», потеряв все переводы курсора на следующую строку.

Изменим эту многострочную строку, добавив после каждого обратного слеша второй обратный слеш (\\). Первый слеш экранирует действие второго слеша, и второй слеш выводится на экран, формируя правый скат крыши. При этом символ перевода курсора на следующую строку не экранируется, поэтому его действие не изменяется (рис. 5).

После вывода видим нормальное отображение домика (рис. 6).

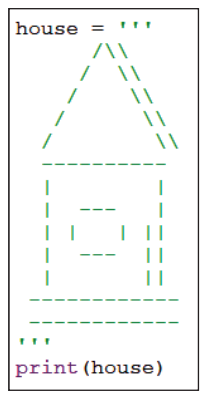

Рис. 5. Текст программы

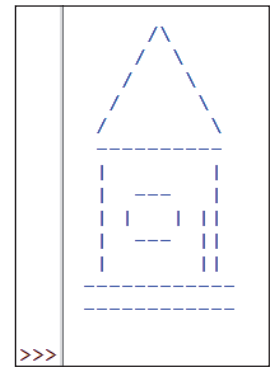

Рис. 6. Правильный вывод многострочной строки «Домик»

ISSN 2221-1993 • **Информатика в школе** • 2025 • Том 24 № 3

```
>>> house
'\n      /\\\n     /  \\\n    /    \\\n   /      \\\n  /\n \\\n  ----------\n  |        |\n  |  ---   |\n  |  ---   |\n  | |   | ||\n  |  ---   ||\n  |        ||\n ------------\n ------------\n'
```

Рис. 7. Внутреннее представление правильной многострочной строки «Домик»

Внутреннее представление строки изменилось (рис. 7).

В качестве практического задания можно предложить учащимся создать многострочную строку со смайликами или произвольными рисунками. Для создания рисунка можно предложить символы клавиатуры и символы псевдографики.

Срезы строк и методы обработки строк [7, 9, 10, 13, 14, 17] предоставляют гибкие инструменты для работы с текстовыми данными в Python. Они позволяют извлекать подстроки, изменять регистр символов, форматировать строки и многое другое. Это делает их незаменимыми при разработке различных приложений и программ, особенно когда требуется работа с большим объемом текстовых данных [6, 12].

2.4. Примеры многострочных строк

На рисунках 8 и 9 представлены примеры многострочных строк, созданных учащимися кружка «Программирование на Python» Донецкого Республиканского Дворца детского и юношеского творчества.

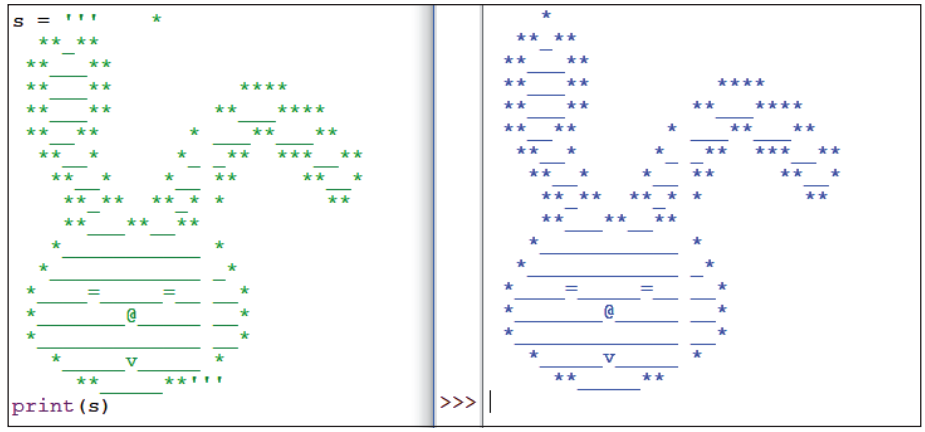

Рис. 8. Программа и результат ее работы, автор Марьяна Ш.

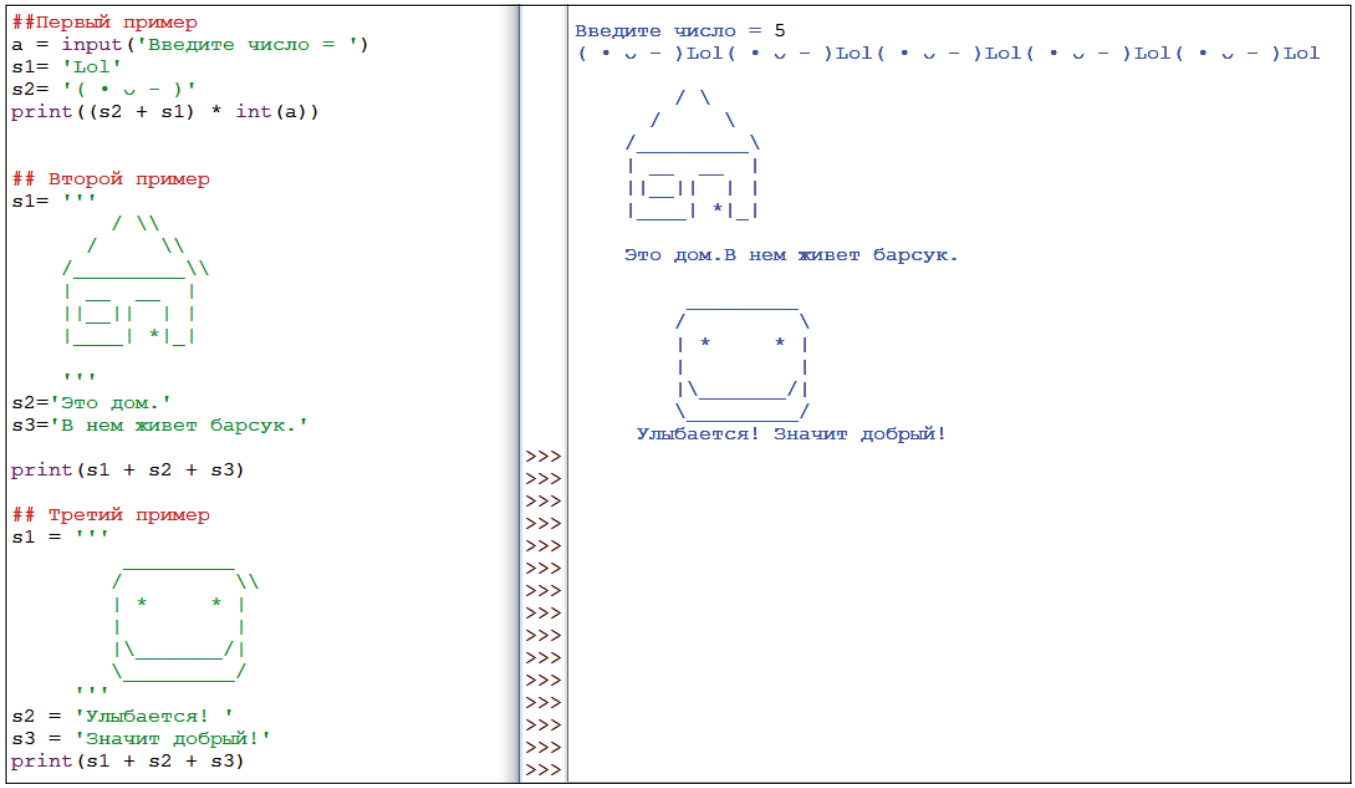

Рис. 9. Программа и результат ее работы, автор Ярослав К.

ISSN 2221-1993 • **Informatics in School** • 2025 • Volume 24 No 3

2.5. Сравнение строк

Строка в Python — это последовательность символов, которая рассматривается как единое целое. Каждый символ в строке имеет свой уникальный код, который соответствует определенному символу в кодовой таблице операционной системы.

Сравнение строк в Python осуществляется с помощью операторов сравнения [14, 15, 18, 19]. В языке программирования Python есть несколько операторов сравнения строк, которые позволяют разработчикам проверять различные условия и принимать решения на основе результатов сравнения. Операторы сравнения строк сравнивают значения строк на основе кодов символов этих строк.

В Python существуют следующие операторы сравнения строк:

- «= =» (равно) — возвращает значение «истина», если две строки имеют одинаковые значения;
- «!=» (не равно) — возвращает значение «истина», если две строки не имеют одинаковых значений;
- «<» (меньше) — сравнивает строки лексикографически, т. е. по порядку символов в алфавите или кодовой таблице;
- «>» (больше) — также сравнивает строки лексикографически;
- «<=» (меньше или равно) — проверяет, что первая строка меньше или равна второй строке;
- «>=» (больше или равно) — проверяет, что первая строка больше или равна второй строке.

При сравнении строк Python учитывает коды символов каждой строки, а не длину строки. Если две строки содержат одинаковые символы в одинаковом порядке, то они считаются равными. Если символы в строках отличаются, то Python сравнивает их коды в соответствии с кодовой таблицей операционной системы. Это позволяет определить, какая строка идет раньше или позже в алфавитном порядке.

Таким образом, сравнение строк в Python основано на сравнении их значений, представленных в виде последовательностей кодов символов из кодовой таблицы операционной системы.

Пример 1.

На рисунке 10 в седьмой строке сравниваются строки ‘1’ и ‘2’. В кодовой таблице код единицы равен 49, а двойки — 50. Коды разные, поэтому результат сравнения дает значение «ложь» (False) (рис. 11).

В восьмой строке сравниваются ‘10’ и ‘11’. Сначала сравниваются первые символы, в обеих строках они одинаковые — это единицы. Поэтому далее сравниваются следующие символы — в одной строке это нуль, а в другой — единица. Код у нуля меньше, поэтому первая строка меньше второй, соответственно, результат сравнения дает значение «истина» (True) (рис. 12).

В девятой строке сравниваются ‘2’ и ‘10’. Результат сравнения — «ложь» (False), так как при сравнении первых символов строк — 2 и 1 — код символа в первой строке больше. Поэтому первая строка больше, а не

```
1  s1 = '1'
2  s2 = '2'
3  s10 = '10'
4  s11 = '11'
5  s100 = '100'
6
7  print(f's1 == s2 - {s1 == s2}')
8  print(f's10 < s11 - {s10 < s11}')
9  print(f's2 < s10 - {s2 < s10}')
10 print(f's11 > s100 - {s11 > s100}')
11 print(f's10 <= s100 - {s10 <= s100}')
12 print(f's2 >= s100 - {s2 >= s100}')
```

Рис. 10. Текст программы сравнения строк примера 1

```
s1 == s2 - False
s10 < s11 - True
s2 < s10 - False
s11 > s100 - True
s10 <= s100 - True
s2 >= s100 - True
>>>
```

Рис. 11. Результат выполнения программы сравнения строк примера 1

Рис. 12. Сравнение строк ‘10’ и ‘11’

Рис. 13. Сравнение строк ‘11’ и ‘100’

ISSN 2221-1993 • **Информатика в школе** • 2025 • Том 24 № 3

```
14 primer = ['ялик', 'ель', 'апельсин', 'ёлка', 'алтарь']
15 print(sorted(primer))
16
```

Рис. 14. Текст программы примера 2

```
>>> ['алтарь', 'апельсин', 'ель', 'ялик', 'ёлка']
```

Рис. 15. Результат выполнения программы примера 2

Рис. 16. Фрагмент таблицы символов операционной системы

меньше второй (несмотря на то, что в числовом представлении 2 < 10).

В десятой строке сравниваются ‘11’ и ‘100’. Результат сравнения — «истина» (True). Код второго символа первой строки (1) больше кода второго символа второй строки (0). Поэтому первая строка больше второй строки (рис. 13).

В одиннадцатой строке сравниваются ‘10’ и ‘100’. При проверке условия первые и вторые символы обеих строк совпадают, а третий символ второй строки не с чем сравнивать, так как в первой строке только два символа. Поэтому вторая строка больше первой. Результат сравнения — «истина» (True).

В двенадцатой строке сравниваются ‘2’ и ‘100’. Первая строка больше второй, так как код первого символа первой строки больше кода первого символа второй строки. Дальнейшее сравнение символов не выполняется, так как результат уже определен. Результат сравнения — «истина» (True).

Пример 2.

Создадим список слов, отсортируем список и выведем на экран (рис. 14).

При выводе получим результат, представленный на рисунке 15.

Первые четыре слова дают вполне предсказуемый результат после сортировки. И только слово «ёлка» выводится в непривычном для нас месте. В алфавите русского языка буква «ё» стоит после буквы «е». Но в таблице символов операционной системы видно, что символ «ё» стоит после всех строчных символов русского языка (рис. 16). Значит, код символа «ё» больше кода символа «я». Поэтому после сортировки слово «ёлка» выведено последним.

3. Заключение

В заключение стоит отметить, что успешное программирование требует не только способности применять уже известные методы и подходы по аналогии с существующими задачами, но и глубокого понимания принципов работы с различными типами данных.

Очень многие задачи предполагают работу с текстовыми данными. И важно иметь детальное представление о строках, их представлении в памяти компьютера и кодовом отображении. Это позволяет эффективно работать со строками, оптимизировать код и избегать возможных ошибок. Такие знания способствуют созданию надежного и гибкого кода, способного адаптироваться к различным условиям и требованиям.

Таким образом, углубленное понимание типа данных «строки» и связанных с ними концепций является необходимым условием для роста программиста и создания качественного программного обеспечения.

Список источников

1. *Бизли Д.* Python. Подробный справочник: пер. с англ. СПб.: Символ-Плюс, 2010. 864 с.

2. *Босова Л. Л.* О новых подходах к изучению школьной информатики в условиях цифровой трансформации общества // Информатика в школе. 2022. № 4 (177). С. 5–14. EDN: DKRLZV. DOI: 10.32517/2221-1993-2022-21-4-5-14 .

3. *Босова Л. Л., Аквилянов Н. А., Кочергин И. О., Штепа Ю. Л., Бурцева Т. А.* Информатика. 8–9 классы. Начала программирования на языке Python. Дополнительные главы к учебникам. М.: БИНОМ. Лаборатория знаний, 2022. 96 с.

4. *Васильев А. Н.* Python на примерах. Практический курс по программированию. СПб.: Наука и Техника, 2016. 432 с.

5. *Горохова Р. И., Догадина Е. П., Долгов В. И., Макрушин С. В.* Практикум по программированию на языке Python: учебное пособие. М.: Финансовый университет, департамент анализа данных и машинного обучения, 2023. 320 с.

6. *Джоши П.* Искусственный интеллект с примерами на Python: пер. с англ. СПб.: ООО «Диалектика», 2019. 448 с.

7. *Доусон М.* Программируем на Python. СПб.: Питер, 2014. 416 с.

8. *Еникеев Р. Р.* Основы и базовые типы данных в Python. Казань: КФУ, 2021. 129 с.

9. *Лутц М.* Изучаем Python: пер. с англ. Ю. Н. Артеменко. СПб.: Диалектика, 2019. 832 с.

10. *Любанович Б.* Простой Python. Современный стиль программирования. СПб.: Питер, 2016. 480 с.

11. *Маркелов В. К., Завьялова О. А.* Возможности языка Python 3.10 при обучении программированию в школе //

ISSN 2221-1993 • **Informatics in School** • 2025 • Volume 24 No 3

Шуйская сессия студентов, аспирантов, педагогов, молодых ученых. Материалы XIV Международной научной конференции, посвященной Году науки и технологий Российской Федерации, 205-летию начала подготовки педагогов в Ивановской области (Москва—Иваново—Шуя, 06–07 октября 2021 года). Шуя: Шуйский филиал Ивановского государственного университета, 2021. С. 96–97. EDN: XXZKLZ.

12. *Маркелов В. К., Завьялова О. А.* Язык программирования Python как альтернативный инструмент для решения заданий ЕГЭ по информатике // Информатика в школе. 2023. № 2 (181). С. 63–72. EDN: UBRWOS. DOI: 10.32517/2221-1993-2023-22-2-63-72.

13. *Поляков К. Ю., Еремин Е. А.* Информатика. 9 класс: учебник. М.: БИНОМ. Лаборатория знаний, 2018. 288 с.

14. *Саммерфильд М.* Программирование на Python 3. Подробное руководство: пер. с англ. СПб.: Символ-Плюс, 2009. 608 с.

15. *Сысоева М. В., Сысоев И. В.* Программирование для «нормальных» с нуля на языке Python: учебник. М.: Базальт СПО; МАКС Пресс. 2018. 176 с.

16. *Федоров Д. Ю.* Программирование на языке высокого уровня Python: учебное пособие для прикладного бакалавриата. М.: Юрайт, 2019. 161 с.

17. *Харрисон М.* Как устроен Python. Гид для разработчиков, программистов и интересующихся: пер. с англ. СПб.: Питер, 2019. 272 с.

18. *Чернышев С. А.* Основы программирования на Python: учебное пособие для вузов. М.: Юрайт. 2022. 286 с.

19. Язык Python // Преподавание, наука и жизнь: сайт Константина Полякова. https://kpolyakov.spb.ru/school/probook/python.htm

20. PEP 8 в Python — правила для идеального кода. М.: Свет Разума, 2013. https://pythonpip.ru/osnovy/pep-8-v-python-pravila-dlya-idealnogo-koda