IT CAREER HUB

Урок 8.

Знакомство со строками

Неизменяемость строк

В Python строки являются неизменяемыми (`immutable`) объектами. Это означает, что после создания строки её содержимое нельзя изменить напрямую. Любое изменение строки приводит к созданию новой строки в памяти, а не изменению исходной. Исходная строка остаётся неизменной.

Пример неизменяемости строки

```python
s = "Hello"

str2 = "мир"

result = str1 + str2

print(result)
```

Конкатенация создаст новую строку, но не изменит изначальную.

"Изменение" содержимого строки:

Если нужно "изменить" строку, это делается через создание новой строки и присваивания её старой переменной

```python
s = "Hello"

s = s + " world!"

print(s)
```

IT CAREER HUB

Задания для закрепления 1

Какой результат будет выведен?

```python
s = "Python"

s += " is fun"

print(s)
```

a. Python
b. Python is fun
c. Ошибка
d. Python is

Посмотреть ответ

IT CAREER HUB

Кодировка

Кодировка — это способ преобразования символов в числа, которые компьютер может хранить и обрабатывать.

Кодировка позволяет компьютерам работать с текстами на разных языках и различными символами.

Кодировка ASCII

ASCII (American Standard Code for Information Interchange) — это одна из первых и простейших кодировок, разработанная для представления текстовых символов в компьютерах.

Она определяет стандартные коды для 128 символов, включая буквы английского алфавита, цифры, символы и управляющие коды.

Основные характеристики кодировки ASCII:

- **7 бит:**
  - ASCII использует 7 бит для кодирования символов.
  - Это позволяет закодировать 128 символов (2^7 = 128).
  - Символы представлены числами от 0 до 127.
- **Символы:**
  - 0-31: Управляющие символы (например, символ новой строки, возврат каретки, сигнал окончания текста).
  - 32-126: Печатные символы, включая буквы, цифры, пунктуацию и другие символы.
- **Простота и совместимость:**
  - ASCII был создан для работы с английским алфавитом, цифрами и основными символами.
  - Он широко используется в компьютерах, сетевых системах и других устройствах, обеспечивая базовый уровень кодирования для текстовых данных.

Таблица символов ASCII

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Таблица символов Unicode

Unicode — это глобальный стандарт, который присваивает каждому символу уникальный числовой код, называемый кодовой точкой.

Он охватывает символы всех языков мира, математические символы, эмодзи и многое другое. Unicode решает проблему ограниченности старых кодировок, таких как ASCII, и позволяет представлять миллионы различных символов.

На данный момент в Unicode определено более 143 000 символов. Это число постоянно растёт, поскольку стандарт Unicode обновляется и добавляются новые символы для различных языков, символов, технических знаков и эмодзи.

Категории символов в Unicode включают:

- Алфавиты различных языков (латинский, кириллица, китайские иероглифы, арабский, и т.д.)
- Математические и технические символы
- Символы для форматирования текста
- Эмодзи
- Диакритические знаки и другие спецсимволы

Unicode разработан так, чтобы поддерживать не только современные, но и исторические языки, а также специализированные символы для научной и технической работы.

Таблицу символов Unicode можно посмотреть на сайте https://symbl.cc/ru/unicode-table/.

Кодировка UTF-8

UTF-8 (Unicode Transformation Format - 8-bit) — это одна из самых популярных и широко используемых кодировок для представления символов в Unicode.

Она преобразует символы Unicode в последовательности байтов переменной длины, где каждый символ может занимать от 1 до 4 байт. UTF-8 стала стандартом для передачи и хранения текстов в Интернете благодаря её гибкости и экономичности.

Как работает кодировка UTF-8:

- **1 байт:** Символы ASCII (от 0 до 127), например, латинские буквы, цифры и некоторые знаки пунктуации.
- **2 байта:** Символы с кодами от 128 до 2047, например, символы латиницы с диакритическими знаками (акценты), кириллические буквы и греческий алфавит.
- **3 байта:** Символы с кодами от 2048 до 65535, включающие китайские иероглифы, японские кандзи и хангыль.
- **4 байта:** Символы с кодами выше 65536, такие как редкие символы и эмодзи.

Преимущества UTF-8:

- **Совместимость:**

    - UTF-8 поддерживает полную совместимость с ASCII, что делает его удобным для интеграции с системами, которые изначально использовали только ASCII.
- **Эффективное использование памяти:**
    - Для текстов на латинице (английский, испанский и т.д.) UTF-8 экономит память, используя только 1 байт на символ.
    - Для текстов на других языках, таких как русский или китайский, UTF-8 остаётся эффективной, занимая меньше памяти по сравнению с UTF-16 или UTF-32.
- **Гибкость:**
    - UTF-8 может кодировать символы любой сложности: от простых букв латинского алфавита до редких символов и эмодзи.
- **Международный стандарт:**
    - UTF-8 стал основным стандартом кодирования текстов в Интернете. Большинство веб-страниц и веб-приложений используют UTF-8 для работы с текстом на разных языках.

IT CAREER HUB

Задания для закрепления 2

1. В чем отличие между кодировками ASCII и UTF-8?

a. UTF-8 поддерживает больше символов, чем ASCII.
b. ASCII поддерживает больше символов, чем UTF-8.
c. В ASCII и UTF-8 одинаковое количество символов.
d. UTF-8 использует 7 бит для кодирования символов.

Посмотреть ответ

2. Какой из следующих диапазонов символов поддерживается кодировкой ASCII?

a. Символы от 0 до 127
b. Символы от 0 до 255
c. Символы от 0 до 1024
d. Символы от 0 до 65535

Посмотреть ответ

3. Какой из вариантов неверен относительно Unicode?

a. Unicode поддерживает символы всех языков мира.
b. В Unicode содержится больше 143 000 символов.
c. Unicode использует 1 бит для кодирования всех символов.
d. Unicode охватывает символы эмодзи и технические знаки.

Посмотреть ответ

IT CAREER HUB

Метод

Метод — это функция, которая принадлежит определённому объекту или классу и может взаимодействовать с его данными.

Основные характеристики метода:

1. **Метод связан с объектом**:
   - Методы вызываются через объекты (экземпляры классов) и могут работать с данными, принадлежащими этим объектам.
   - Например, строка в Python является объектом класса str, и у неё есть методы для работы с текстом.
2. **Вызов метода**:
   - Чтобы вызвать метод, необходимо использовать синтаксис: `object_name.method_name`
   - (). Например, для строкового объекта это может быть так:

```python
text = "hello"
print(text.encode('utf-8'))  # Кодирует текст, полученный из переменной text в 'utf-8'
```

3. **Методы могут принимать аргументы**:
   - Как и функции, методы могут принимать аргументы, которые помогают управлять их поведением.

Методы encode и decode

В Python для работы с кодировками используются методы encode() и decode():

- `encode()` преобразует строку в байты, используя указанную кодировку (например, UTF-8). Это полезно при передаче данных по сети или записи в файл.
- `decode()` делает обратное — преобразует байты обратно в строку, интерпретируя их как символы на основе кодировки.

Пример кодирования строки в UTF-8:

```python
text = "Python"
# Метод .encode() кодирует строку в байты с использованием UTF-8
encoded_text = text.encode('utf-8')
print(encoded_text)

# Оригинальная строка
text = "Привет"

# Кодирование строки в байты с использованием UTF-8
encoded_text = text.encode('utf-8')
print(encoded_text)
```

Декодирование строки обратно в текст:

```python
# Декодирование байтов обратно в строку
decoded_text = encoded_text.decode('utf-8')
print(decoded_text)
```

Где нужны кодировки:

1. **Работа с файлами**: При чтении и записи файлов часто необходимо указывать кодировку, особенно если файлы содержат символы, отличные от ASCII.
2. **Работа с сетевыми данными**: Данные, передаваемые через сеть, могут быть закодированы, и для их правильной обработки необходимо правильно декодировать.
3. **Международные приложения**: Если ваше приложение работает с разными языками, важно использовать кодировки, поддерживающие многоязычные символы, например, **UTF-8**.

Функции ord, chr

ord() и chr() — это встроенные функции Python, которые работают с символами и их кодировкой.

Они позволяют конвертировать символы в их числовое представление (код Unicode) и наоборот.

Функция ord()

Функция `ord()` принимает символ и возвращает его код Unicode. Это полезно, когда нужно узнать числовое значение символа.

Пример

```python
print(ord('A'))
print(ord('a'))
print(ord(' '))
```

Функция chr()

Функция `chr()` принимает **число**, представляющее код Unicode, и возвращает соответствующий **символ**.

Пример

```python
print(chr(65))
print(chr(97))
print(chr(32))
```

IT CAREER HUB

Задания для закрепления 3

1. Что делает функция ord() в Python?

a. Преобразует строку в байты
b. Возвращает числовой код символа Unicode
c. Преобразует число в строку
d. Возвращает кодировку символа

Посмотреть ответ

2. Какой метод используется для преобразования строки в байты с указанием кодировки?

a. `decode()`
b. `ord()`
c. `encode()`
d. `chr()`

Посмотреть ответ

Сравнение строк

В Python строки можно сравнивать с помощью операторов сравнения, таких как ==, !=, <, >, <= и >=.

Сравнение строк происходит на основе лексикографического порядка, что означает, что строки сравниваются по символам с использованием их кода в Unicode (например, 'A' < 'B', 'a' > 'A').
В результате сравнения будет получено логическое значение True или False.

Примеры сравнения строк

1. **Оператор ==** — проверяет равенство двух строк:

```python
print("apple" == "apple")

print("apple" == "banana")
```

2. **Оператор !=** — проверяет неравенство двух строк:

```python
print("apple" != "ap ple")
```

3. **Операторы <, >, <=, >=** — сравнивают строки лексикографически. Строки сравниваются символ за символом на основе их Unicode кодов. (Буквы сравниваются с учётом регистра, т.к. имеют разный код).

```python
print("apple" < "banana")   # True('a' < 'b')

print("Apple" < "banana")   # True ('A' < 'b')
```

```
print("apple" > "Apple")    # True ('a' > 'A')
```

Как происходит сравнение:

- **Лексикографический порядок:** Python сравнивает строки символ за символом. Как только найдено различие между двумя символами, сравнение завершается.

```
Python
print("abc" < "aca")  # True, так как 'b' < 'c'
```

- **Учёт регистра:** В Unicode строчные буквы (например, 'a') имеют больший код, чем заглавные (например, 'A'). Поэтому заглавные буквы считаются "меньшими" по значению.

```
Python
print("apple" > "Apple")  # True, так как 'a' > 'A'
```

- **Сравнение строк разной длины:** Когда строки разной длины сравниваются, Python продолжает сравнение, пока не закончатся символы в одной из строк. Строка, у которой символы закончились первой, считается "меньшей".

```
Python
print("apple" < "app")  # False, так как "apple" не короче
```

Функция len

Функция len() используется для получения длины объектов, таких как строки и другие последовательности. Она возвращает количество элементов в объекте.

Пример использования функции len() для строк

В случае строки функция len() возвращает количество символов в строке.

```python
text = "Python"

length = len(text)

print(length)
```

IT CAREER HUB

Задания для закрепления 4

1. Что произойдет, если сравнить строки разной длины?

```python
print("cat" > "catalog")
```

a. `True`
b. `False`
c. Ошибка
d. Невозможно сравнить строки

Посмотреть ответ

2. Какой результат будет выведен при сравнении строк с учётом регистра?

```python
print("Zoo" < "zoo")
```

a. `True`
b. `False`
c. Ошибка
d. Невозможно сравнить строки

Посмотреть ответ

3. Какой результат будет выведен при сравнении строк с пробелом?

```python
print("apple" != "ap ple")
```

a. `True`
b. `False`
c. Ошибка

d. Невозможно сравнить строки

Посмотреть ответ

4. Какая функция возвращает длину строки?

a. `length()`
b. `len()`
c. `size()`
d. `count()`

Посмотреть ответ

Индексация строк

Индексация — это механизм обращения к отдельным элементам в последовательных структурах данных, таких как строки и другие.

В Python индексация начинается с 0 (а не с единицы, как мы привыкли считать), что означает, что первый элемент последовательности имеет индекс 0, второй — индекс 1, и так далее.

Пример индексации

```python
Для строк:

text = "Python"

print(text[0])   # Вывод: 'P' (первый символ)

print(text[2])   # Вывод: 't' (третий символ)
```

Отрицательная индексация

В Python можно использовать отрицательные индексы для обращения к элементам с конца последовательности. Последний элемент имеет индекс -1, предпоследний — -2 и так далее.

Пример

```python
text = "Python"

print(text[-1])  # Вывод: 'n' (последний символ)
```

```
print(text[-3])  # Вывод: 'h' (третий символ с конца)
```

Ошибка при обращении по несуществующему индексу

Если вы попытаетесь обратиться к символу по индексу, который выходит за пределы длины строки, Python сообщит об ошибке.

Пример ошибки

```
Python
text = "hello"

print(text[10])  # Ошибка, так как в строке "hello" всего 5 символов

print(text[-6])  # Ошибка, так как в строке "hello" всего 5 символов
```

```
text = "hello"
print(text[10])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[1], line 2
      1 text = "hello"
----> 2 print(text[10])

IndexError: string index out of range
```

Как избежать ошибки:

1. **Проверять длину строки:** Прежде чем обращаться к символу по индексу, можно убедиться, что этот индекс находится в пределах допустимой длины строки. Для этого используется функция len(), которая возвращает длину строки.

```python
text = "hello"

index = 5



if index < len(text):

    print(text[index])

else:

    print("Индекс вне диапазона!")
```

Срезы строк

Срезы — это способ получения подстрок из существующей строки, используя индексы. В Python срезы позволяют выбрать часть строки, указав начальный и конечный индексы.

При этом начальный индекс включается в срез, а конечный — не включается.

Синтаксис срезов:

```python
строка[start:end]
```

- **start** — индекс, с которого начинается срез (включительно).
- **end** — индекс, на котором срез заканчивается (не включительно).

Если не указывать `start` или `end`, Python будет считать, что срез начинается с начала строки или заканчивается в конце строки соответственно.

Пример среза строки

```python
text = "Python programming"

print(text[0:6])
```

В этом примере мы взяли символы с индексами от 0 до 5 включительно, что даёт подстроку "Python".

Срезы без указания индексов

- Если не указывать начальный индекс, срез начнётся с начала строки:

```python
text = "Python programming"

print(text[:6])
```

- Если не указывать конечный индекс, срез продлится до конца строки:

```python
text = "Python programming"

print(text[7:])
```

- Если не указывать оба индекса, срез создаст копию всей строки:

```python
text = "Python programming"

print(text[:])
```

Отрицательные индексы

Можно использовать отрицательные индексы для срезов с конца строки. Например, индекс -1 соответствует последнему символу строки.

Пример

```python
text = "Python programming"

print(text[-11:])
```

Порядок индексов

При указании индекса нужно учитывать что end должен быть больше чем start.

Пример

```python
text = "Python programming"

print(text[-11:-4])  # -11 < -4

print(text[7:14])  # 7 < 14
```

При указании некорректных индексов результатом будет пустая строка.

Пример

```python
text = "Python programming"

print(text[-4:-11])  # Пустая строка

print(text[14:7])  # Пустая строка
```

Срезы с шагом

Срезы в Python можно использовать не только индексами, но и с шагом (`step`), который указывает, через сколько символов выбирать элементы. Шаг по умолчанию равен 1, но его можно изменить, задав положительное или отрицательное значение.

Синтаксис с шагом

```python
строка[start:end:step]
```

- **start** — начальный индекс (включительно).
- **end** — конечный индекс (не включительно).
- **step** — шаг, который определяет, через сколько символов брать элементы. Если шаг отрицательный, строка будет срезаться в обратном порядке.

Пример с положительным шагом

```python
text = "Python programming"
print(text[0:12:2])  # Вывод: 'Pto rg'
```

Здесь шаг равен 2, поэтому берутся каждый второй символ из строки "Python prog".

Обратный шаг

Шаг может быть отрицательным, что позволяет создавать срез строки в обратном порядке.

Пример с отрицательным шагом

```python
text = "Python programming"
print(text[12:6:-1])  # Вывод: 'argorp'
```

Здесь срез начинается с символа с индексом 12 и идёт до символа с индексом 7 (не включительно), двигаясь в обратном порядке с шагом -1.

Порядок индексов при отрицательном шаге

При указании отрицательного шага нужно учитывать что теперь end должен быть меньше чем start, т.к. выбор символов происходит в обратном порядке.

Пример

```python
text = "Python programming"
print(text[-4:-11:-1])  # -11 < -4
print(text[14:7:-1])  # 7 < 14
```

При указании некорректных индексов результатом будет пустая строка.

Пример

```python
text = "Python programming"
print(text[-11:-4:-1])  # Пустая строка
print(text[7:14:-1])  # Пустая строка
```

Срез всей строки в обратном порядке

Если указать шаг -1, можно развернуть всю строку:

```python
text = "Python"
print(text[::-1])  # Вывод: 'nohtyP'
```

Оператор принадлежности in

Оператор in в Python используется для проверки, содержит ли строка определённый элемент или подстроку.

Если элемент найден, оператор возвращает `True`, если нет — `False`.

Пример с использованием оператора in для строк

```python
text = "Python programming"

print("Python" in text)

print("Hello" in text)
```

Комбинация not и in:

Комбинация `not in` используется для проверки, что элемент не содержится в строке или другой последовательности. not просто инвертирует результат выражения с оператором `in`.

Пример

```python
text = "Python programming"

print("Java" not in text)
```

IT CAREER HUB

Задания для закрепления 5

1. Какой результат будет выведен?

```python
text = "Hello, Python!"
print(text[7])
```

a. H
b. o
c. P
d. Ошибка

2. Какой результат будет выведен?

```python
text = "Hello, Python!"
print(text[-3])
```

a. t
b. o
c. h
d. o

3. Какой результат будет выведен?

```python
text = "Hello, Python!"
print(text[1:6])
```

a. `Hello`
b. `ello,`
c. `ello`
d. Ошибка

4. Какой результат будет выведен?

```python
text = "Hello"
print(text[10])
```

a. Вывод символа с индексом 10
b. Вывод первого символа
c. Ошибка
d. Вывод последнего символа

5. Какой результат будет выведен?

```python
text = "Python"
print(text[::-1])
```

a. Python
b. nohtyP
c. nohty
d. Ошибка

6. Какой результат будет выведен?

```python
text = "Python programming"
print("python" in text)
```

a. True
b. False
c. Ошибка
d. None

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
Результат выполнения кода	Ответ: b
Задания на закрепление 2	Вернуться к заданиям
1. Отличия кодировок	Ответ: a
2. Диапазон символов кодировки ASCII	Ответ: a
3. Утверждения о Unicode	Ответ: c
Задания на закрепление 3	Вернуться к заданиям
1. Функция `ord()`	Ответ: b
2. Преобразование строки в байты с указанием кодировки	Ответ: c
Задания на закрепление 4	Вернуться к заданиям
1. Сравнение строк разной длины	Ответ: b
2. Сравнение строк с учётом регистра	Ответ: a
3. Сравнение с пробелом	Ответ: a
4. Длина строки	Ответ: b
Задания на закрепление 5	Вернуться к заданиям
1. Результат выполнения кода	Ответ: c
2. Результат выполнения кода	Ответ: d
3. Результат выполнения кода	Ответ: c
4. Результат выполнения кода	Ответ: c
5. Результат выполнения кода	Ответ: b
6. Результат выполнения кода	Ответ: b

IT CAREER HUB

Практические задания

1. Палиндром

Напишите программу, которая проверяет является ли строка палиндромом. Палиндром - это строка, которая читается одинаково слева направо и справа налево).

Пример вывода:

```
Python
Введите строку: radar
Строка является палиндромом
```

Решение:

```python
text = input("Введите строку: ")

if text == text[::-1]:
    print("Строка является палиндромом")
else:
    print("Строка не является палиндромом")
```

2. Шифр Цезаря

Напишите программу для дешифровки текста, зашифрованного с помощью шифра Цезаря. Программа принимает зашифрованный текст и сдвиг (использованный для зашифровки) от пользователя и выводит расшифрованный текст.

Шифр Цезаря — это один из самых простых и известных методов шифрования. В этом методе каждый символ в исходном тексте заменяется на символ, находящийся на фиксированное число позиций дальше в алфавите. Сдвиг может быть как влево, так и вправо, и для расшифровки необходимо просто сдвинуть символы обратно на тот же шаг.

Пример вывода:

```python
Введите зашифрованный текст: khoor
Введите сдвиг: 3
Расшифрованный текст: hello
```

Решение:

```python
encrypted_text = input("Введите зашифрованный текст: ")
shift = int(input("Введите сдвиг, использованный для зашифровки: "))

decrypted_text = ''

i = 0
while i < len(encrypted_text):
    char = encrypted_text[i]
    # Преобразуем символ в код, сдвигаем обратно, и преобразуем обратно в
символ
    decrypted_char = chr(ord(char) - shift)
    decrypted_text += decrypted_char
    i += 1

print("Расшифрованный текст:", decrypted_text)
```