КУРС «ИНФОРМАТИКА»

Часть 1. Символьные переменные и строки

2021 – 2022 УЧЕБНЫЙ ГОД

Символьный тип Char (Pascal, Java, C…)

Переменная типа char -1 байт (ASCII) $\Rightarrow$ 2^8 = 256 возможных значений

Переменная типа char - 2 байта (Unicode) $\Rightarrow$ 2^{16} = 65536 возможных значений

Фрагмент таблицы ASCII - кодов

32	пробел
33	!
34	"
35	#
36	$
37	%
38	&
39	'
40	(
41	)

48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9

65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J

97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j

Тип char – это тип данных, служащий для хранения одиночных символов в различных кодировках.
Он широко используется в таких языках программирования, таких как Pascal, Java, C.
Даже строки там являются массивами, состоящими из элементов типа char.

Строковый тип String (Python)

В Python нет отдельного типа для символов. Даже если присвоить переменной значение ‘a’, она будет иметь строковый тип.
Строковый тип предоставляет программисту весь нужный функционал для работы как со строками, так и с символами.
Любой символ в Python является единичной строкой, что позволяет использовать для работы с ним те же функции, что и для строк.

Строка — это неизменяемая последовательность!

```
s = "ABCD"
s = "F"
print(s)

F
```

```
s = "A"
s[0] = "F"
print(s)

s[0] = "F"
TypeError: 'str' object does not support item assignment
```

Функции работы с символами

Обращение к символу: “A” или ‘A’

Сравнение символов: ‘A’<‘B’, ‘0’>’!’, ‘z’=‘z’, ‘8’<>’*’

`ord(ch)` – выдает номер символа (нумерация с нуля):

```
print(ord('0'));  // выводит 48
```

`chr(k)` – выдает k-ый символ из таблицы символов

```
print(chr(71));  // выводит символ “G”
```

Экранирование

Экранированные символы (escape - последовательности) — это специальные символы после обратной косой черты «\», выполняющие определенные действия и преобразования

Экранированная последовательность	Функция
\n	Переход на новую строку
\t	Табуляция
\r	Установка курсора в начало строки
\x	Числа в шестнадцатеричном представлении
\o	Числа в восьмеричном представлении
\0	Нулевой символ
\’	Апостроф
\»	Двойная кавычка
\\	Обратный слэш

```
s = "Hello,\n\"World\""
print(s)



#Выведет:
Hello,
"World"
```

Подавление экранирования

1. При работе с путями к файлам необходимо использовать сразу два слэша

```
s = "C:\\Users\\Public"
print(s)

#Выведет:
C:\Users\Public
```

2. Подавить экранирование можно с помощью «r», который ставится перед началом строки (до кавычек). При этом интерпретатор, видя перед строкой «r», автоматически дублирует каждый символ обратного слеша

```
s = r"C:\Users\Public"
print(s)

#Выведет:
C:\Users\Public
```

Строка, в которой подавляется экранирование, **не может оканчиваться** символом обратного слеша!
В противном случае интерпретатор вызовет исключение SyntaxError.

КУРС «ИНФОРМАТИКА»

Часть 2. Строковый тип str

2021 – 2022 УЧЕБНЫЙ ГОД

Строковый тип str

Строка в Python - упорядоченная **неизменяемая** последовательность символов, используемая для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме

Размерность и состав созданной однажды неизменяемой последовательности **не может меняться**,
вместо этого обычно **создаётся новая последовательность.**

Строка заключена в кавычки

Создание/ввод строк

1. С помощью одинарных и двойных кавычек. Строки в одинарных и двойных кавычках - одно и то же. Причина наличия двух вариантов в том, чтобы позволить вставлять в строки символы кавычек, не используя экранирование

```
s1 = 'Текст в одинарных кавычках'
s2 = "Текст в двойных кавычках"
s3 = 'Слово "Сессия" обычно подразумевает проблемы'
s4 = "I'm learning Python'
```

2. С помощью тройных кавычек.
Тройные кавычки можно использовать для записи многострочных блоков текста. Внутри такой строки возможно присутствие кавычек и апострофов, главное, чтобы не было трех кавычек подряд.

```
s5 = '''Это очень длинная
       строка, ей нужно
       много места'''
```

3. С помощью ввода

```
s6 = input()
```

4. С помощью ввода из файла

```
f = open('test.txt')
s7 = f.readline()
```

Методы работы со строками

Метод	Описание	Пример

len(s)	Длина последовательности	`s = "математика"`
		`print(len(s))`
		`# выведет 10`

x in s	Если элемент присутствует в последовательности, то возвращает True, иначе - False	`# Количество гласных в строке s`
		`k = 0`
		`for c in s:`
		`  if c in "AEIOYU":`
		`  k+=1`

s + t	Конкатенация(сложение) двух последовательностей	`s = "Скоро "`
		`t = "сессия"`
		`print( s + t) #s=«Скоро сессия"`

s * n	Эквивалентно сложению последовательности s с собой n раз	`s = "w"*10` `print(s) #s="wwwwwwwwww"`

Методы работы со строками

Метод	Описание	Пример

min(s)	Минимальный элемент последовательности	`s = "математика"`
		`print(min(s))`
		`# выведет «а»`

max(s)	Максимальный элемент последовательности	`s = "математика"`
		`print(max(s))`
		`# выведет «т»`

s.index(x)	Возвращает индекс подстроки x в строке s	`s = "математика"`
		`print(s.index("и"))`
		`# выведет 7`

s.count(x)	Число вхождений подстроки x в строку s	`s = "математика"`
		`print(s.count("а"))`
		`# выведет 3`

s[i]	Возвращает i-й элемент последовательности – срез с одним параметром	`s = input()`
		`for i in range (len(s)):`
		`  if s[i]==s[i+1]...`

s[i, j]	Возвращает набор элементов последовательности с индексами из диапазона i <= k < j - срез с двумя параметрами	`s = "математика"`
		`s = s[2:6]`
		`# s = "тема"`

Срезы

Срез (slice) — извлечение из данной строки одного символа или некоторого фрагмента подстроки или подпоследовательности.

Индекс - номер символа в строке (а также в других структурах данных: списках, кортежах).

Нумерация начинается с **0**.

Если указать отрицательное значение индекса, то номер будет отсчитываться с конца, начиная с номера **-1**.

Любые операции среза со строкой **создают новые строки** и **никогда не меняют исходную строку**.

Срезы

```
>>> str = 'Hello'
```

1. Срез с одним параметром - взятие одного символа строки

```
>>> str[0]   'H‘
```

2. Срез с двумя параметрами `s[a:b]` возвращает подстроку, начиная с символа с индексом **a** до символа с индексом **b**, не включая его. Если опустить второй параметр (но поставить двоеточие), то срез берется до конца строки.

```
>>> str[0:4] 'Hell'
>>> str[0:5] 'Hello'
>>> str[1:3] 'el'
>>> str[1:]  'ello'
>>> str[0:]  'Hello'
```

Срезы

```
>>> str = 'Hello'
```

3. Срез с тремя параметрами - `s[a:b:d]` .
Третий параметр задает шаг(как в случае с функцией **range**), то есть будут взяты символы с индексами **a, a + d, a + 2 * d** и т. д.

Например, при задании значения третьего параметра, равному **2**, в срез попадет каждый второй символ

```
>>> str[0:5:1] 'Hello'
>>> str[::1]   'Hello'
>>> str[0:5:2] 'Hlo'
>>> str[::2]   'Hlo'
```

Примеры работы с символами и строками

```
# сформировать строку с заглавными латинскими буквами
 lat=""
for i in range(ord("A"),ord("Z")+1):
    lat=lat + chr(i)

# вывести на экран таблицу ASCII – кодов

for i in range (256):
    print(i, chr(i))

# организовать «бесконечный» цикл
s = "*"
while s!="exit":
    print("Для выхода из цикла введите exit")
    s =  input()
```

КУРС «ИНФОРМАТИКА»

Часть 3. Файлы в Python

2020 – 2021 УЧЕБНЫЙ ГОД

Файлы в Python

Бинарные файлы

В бинарных файлах данные отображаются в закодированной форме (с использованием только нулей (0) и единиц (1) вместо простых символов).
В большинстве случаев это просто последовательности битов.
Они хранятся в формате `.bin`.

Текстовые файлы

В них хранятся последовательности символов. Блокнот и другие стандартные редакторы умеют читать и редактировать этот тип файлов.
Текст может храниться в двух форматах: `(.txt)` — простой текст и `(.rtf)` — «формат обогащенного текста».

КУРС «ИНФОРМАТИКА»

Часть 3. Текстовые файлы

2020 – 2021 УЧЕБНЫЙ ГОД

Текстовые файлы

Текстовый файл – это файл, в котором:
а) информация представлена в текстовом виде посредством символов из таблицы ASCII - кодов;
б) информация может разделяться на строки произвольной длины, которые разделены между собой метками конца строки (#10#13)
в) весь файл заканчивается меткой конца файла (#26)
г) при записи чисел, строк и логических значений они преобразуются в символьный (текстовый вид)

Действия с файлом:
1. Открыть
2. Записать/Прочитать
3. Закрыть

Открытие/закрытие файла

1 способ — использование метода **open:**

```
f = open('test.txt')
```

после окончания работы с файлом его необходимо принудительно закрыть:

```
f.close()
```

2 способ — использование метода **with**, которая упрощает обработку исключений с помощью инкапсуляции начальных операций, а также задач по закрытию и очистке.
В таком случае инструкция close не нужна, потому что with автоматически закроет файл.

```
with open('test.txt') as f:
    # работа с файлом
```

Метод open()

```
f = open(file_name, access_mode)
```

где **file_name** = имя открываемого файла

access_mode = режим открытия файла. Он может быть: для чтения, записи и т. д.

По умолчанию используется режим чтения (r), если другое не указано.

Режим	Описание	Режим	Описание
r	Только для чтения.	w+	Для чтения и записи. Создаст новый файл для записи, если не найдет с указанным именем.
w	Только для записи. Создаст новый файл, если не найдет с указанным именем.	wb+	Для чтения и записи (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
rb	Только для чтения (бинарный).	a	Откроет для добавления нового содержимого. Создаст новый файл для записи, если не найдет с указанным именем.
wb	Только для записи (бинарный). Создаст новый файл, если не найдет с указанным именем.	a+	Откроет для добавления нового содержимого. Создаст новый файл для чтения записи, если не найдет с указанным именем.
r+	Для чтения и записи.	ab	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для записи, если не найдет с указанным именем.
rb+	Для чтения и записи (бинарный).	ab+	Откроет для добавления нового содержимого (бинарный). Создаст новый файл для чтения записи, если не найдет с указанным именем.

```
f = open('example.txt','r')     # открыть файл из рабочей папки в режиме чтения
fp = open('c:/test.txt','r')    # открыть файл из любого каталога
```

Закрытие файла

1 способ — использование метода **close**()

```
>>> f.close() # закрыть файл
```

2 способ — использование конструкции **try/finally**, которая гарантирует, что если после открытия файла операции с ним приводят к исключениям, он закроется автоматически. Без нее программа завершается некорректно.

```
>>> f = open('example.txt','r')
>>> try:
>>>     # работа с файлом
>>> finally:
>>>     f.close()
```

Файл нужно открыть до инструкции **try**, потому что если инструкция **open** сама по себе вызовет ошибку, то файл не будет открываться для последующего закрытия.
Этот метод гарантирует, что если операции над файлом вызовут исключения, то он закроется до того как программа остановится.