

Типы и модель данных

С.В. Лемешевский (sergey.lemeshevsky@gmail.com)

Институт математики НАН Беларуси

Mar 18, 2020

Содержание

1	Кратко о типизации языков программирования	2
2	Типы данных в Python	2
2.1	Модель данных	3
3	Изменяемые и неизменяемые типы данных	6
4	Целочисленные типы	8
4.1	Логические значения	11
5	Типы чисел с плавающей точкой	11
5.1	Числа с плавающей точкой	12
5.2	Комплексные числа	14
5.3	Числа типа <code>Decimal</code>	15
6	Строки	16
6.1	Сравнение строк	17
6.2	Получение срезов строк	18
6.3	Операторы и методы строк	20
6.4	Форматирование строк с помощью метода <code>str.format()</code>	21
7	Примеры	22
7.1	Печать символов Юникода	22
7.2	Решение квадратного уравнения	25
7.3	Представление таблицы <code>csv</code> в HTML	27

8 Упражнения	31
8.1 Изменение вывода символов Юникода	31
8.2 Изменение <code>quadratic.py</code>	32
8.3 Использование функции <code>escape()</code>	32
8.4 Добавление обработки параметров командной строки в <code>csv2html.py</code>	32

Здесь разберем как Python работает с переменными и определим, какие типы данных можно использовать в рамках этого языка. Подробно рассмотрим модель данных Python, а также механизмы создания и изменения значения переменных.

1. Кратко о типизации языков программирования

Если достаточно формально подходить к вопросу о типизации языка Python, то можно сказать, что он относится к языкам с неявной сильной динамической типизацией.

Неявная типизация означает, что при объявлении переменной вам не нужно указывать её тип, при явной – это делать необходимо. В качестве примера языков с явной типизацией можно привести Java, C++. Вот как будет выглядеть объявление целочисленной переменной в Java и Python.

- Java:

```
int a = 1 ;
```

- Python:

```
a = 1
```

2. Типы данных в Python

В Python типы данных можно разделить на встроенные в интерпретатор (**built-in**) и не встроенные, которые можно использовать при импортировании соответствующих модулей.

К основным встроенным типам относятся:

1. None (неопределенное значение переменной)
2. Логические переменные (**Boolean Type**)
3. Числа (**Numeric Type**)
 - (a) `int` – целое число
 - (b) `float` – число с плавающей точкой

- (c) `complex` – комплексное число
- 4. Списки (Sequence Type)
 - (a) `list` – список
 - (b) `tuple` – кортеж
 - (c) `range` – диапазон
- 5. Строки (Text Sequence Type)
 - (a) `str`
- 6. Бинарные списки (Binary Sequence Types)
 - (a) `bytes` – байты
 - (b) `bytearray` – массивы байт
 - (c) `memoryview` – специальные объекты для доступа к внутренним данным объекта через `protocol buffer`
- 7. Множества (Set Types)
 - (a) `set` – множество
 - (b) `frozenset` – неизменяемое множество
- 8. Словари (Mapping Types)
 - (a) `dict` – словарь

2.1. Модель данных

Рассмотрим как создаются объекты в памяти, их устройство, процесс объявления новых переменных и работу операции присваивания.

Для того, чтобы объявить и сразу инициализировать переменную необходимо написать её имя, потом поставить знак равенства и значение, с которым эта переменная будет создана.

Например строка:

```
b = 5
```

Объявляет переменную `b` и присваивает ей значение 5.

Целочисленное значение 5 в рамках языка Python по сути своей является *объектом*. Объект, в данном случае – это абстракция для представления данных, данные – это числа, списки, строки и т.п. При этом, под *данными* следует понимать как непосредственно сами объекты, так и отношения между ними (об этом чуть позже). Каждый объект имеет три атрибута – это *идентификатор*, *значение* и *тип*.

Идентификатор – это уникальный признак объекта, позволяющий отличать объекты друг от друга, а *значение* – непосредственно информация, хранящаяся в памяти, которой управляет интерпретатор.

При инициализации переменной, на уровне интерпретатора, происходит следующее:

- создается целочисленный объект 5 (можно представить, что в этот момент создается ячейка и число 5 «кладется» в эту ячейку);
- данный объект имеет некоторый идентификатор, значение: 5, и тип: целое число;
- посредством оператора = создается ссылка между переменной b и целочисленным объектом 5 (переменная b ссылается на объект 5).



Об именах переменных

Допустимые имена переменных в языке Python – это последовательность символов произвольной длины, содержащей «начальный символ» и ноль или более «символов продолжения». Имя переменной должно следовать определенным правилам и соглашениям.

Первое правило касается начального символа и символов продолжения. Начальным символом может быть любой символ, который в кодировке Юникод рассматривается как принадлежащий диапазону алфавитных символов ASCII (a, b, ..., z, A, B, ..., Z), символ подчеркивания (_), а также символы большинства национальных (не английских) алфавитов. Каждый символ продолжения может быть любым символом из тех, что пригодны в качестве начального символа, а также любым непробельным символом, включая символы, которые в кодировке Юникод считаются цифрами, такие как (0, 1, ..., 9), и символ Катаkana `·`. Идентификаторы чувствительны к регистру, поэтому TAXRATE, Taxrate, TaxRate, taxRate и taxrate – это пять разных переменных.

Имя переменной не должно совпадать с ключевыми словами интерпретатора Python. Список ключевых слов можно получить непосредственно в программе, для этого нужно подключить модуль `keyword` и воспользоваться командой `keyword.kwlist`.

```
import keyword
print("Python keywords: " , keyword.kwlist)
```

Проверить является ли идентификатор ключевым словом можно так:

```
>>> keyword.iskeyword("try")
True
```

```
>>> keyword.iskeyword( "b" )
False
```



Об использовании символа подчеркивания в именах переменных

Не должны использоваться имена, начинающиеся и заканчивающиеся двумя символами подчеркивания (такие как `__lt__`). В языке Python определено множество различных специальных методов и переменных с такими именами (и в

случае специальных методов мы можем заменять их, то есть создать свои версии этих методов), но мы не должны вводить новые имена такого рода.

Символ подчеркивания сам по себе может использоваться в качестве идентификатора; внутри интерактивной оболочки интерпретатора или в командной оболочке Python в переменной с именем `_` сохраняется результат последнего вычисленного выражения. Во время выполнения обычной программы идентификатор `_` отсутствует, если мы явно не определяем его в своем программном коде. Некоторые программисты любят использовать `_` в качестве идентификатора переменной цикла в циклах `for ... in`, когда не требуется обращаться к элементам, по которым выполняются итерации. Например:

```
for _ in (0, 1, 2, 3, 4, 5):  
    print("Hello")
```

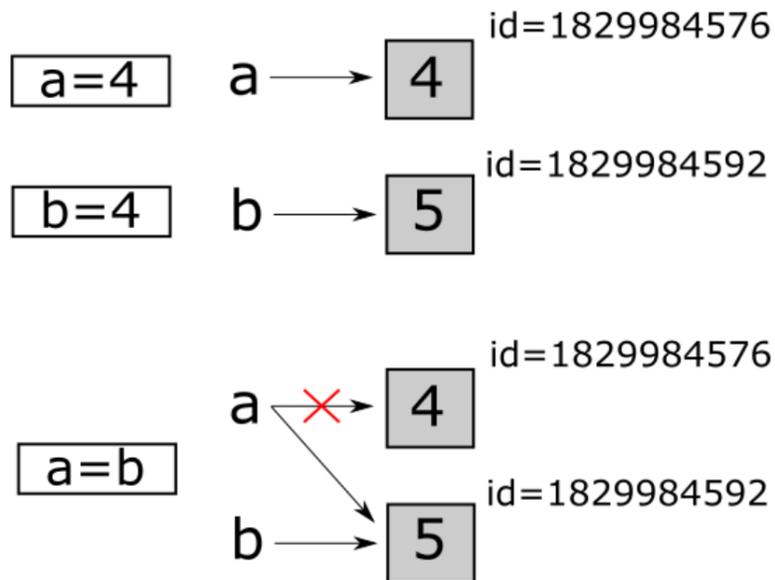
Для того, чтобы посмотреть на объект с каким идентификатором ссылается данная переменная, можно использовать функцию `id()`.

```
>>> a = 4  
>>> b = 5  
>>> id(a)  
1829984576
```

```
>>> id(b)  
1829984592
```

```
>>> a = b  
>>> id(a)  
1829984592
```

Как видно из примера, идентификатор – это некоторое целочисленное значение, посредством которого уникально адресуется объект. Изначально переменная `a` ссылается на объект `4` с идентификатором `1829984576`, переменная `b` – на объект с `id = 1829984592`. После выполнения операции присваивания `a = b`, переменная `a` стала ссылаться на тот же объект, что и `b`.



Тип переменной можно определить с помощью функции `type()`. Пример использования приведен ниже.

```
>>> a = 10
>>> b = "hello"
>>> c = ( 1 , 2 )
>>> type (a)
< class 'int' >
```

```
>>> type (b)
< class 'str' >
```

```
>>> type (c)
< class 'tuple' >
```

3. Изменяемые и неизменяемые типы данных

В Python существуют изменяемые и неизменяемые типы.

К неизменяемым (`immutable`) типам относятся:

- целые числа (`int`);
- числа с плавающей точкой (`float`);

- комплексные числа (`complex`);
- логические переменные (`bool`);
- кортежи (`tuple`);
- строки (`str`);
- неизменяемые множества (`frozen set`).

К изменяемым (`mutable`) типам относятся

- списки (`list`);
- множества (`set`);
- словари (`dict`).

Как уже было сказано ранее, при создании переменной, вначале создается объект, который имеет уникальный идентификатор, тип и значение, после этого переменная может ссылаться на созданный объект.

Неизменяемость типа данных означает, что созданный объект больше не изменяется. Например, если мы объявим переменную `k = 15`, то будет создан объект со значением `15`, типа `int` и идентификатором, который можно узнать с помощью функции `id()`.

```
>>> k = 15
>>> id(k)
1672501744
```

```
>>> type(k)
< class 'int' >
```

Объект с `id = 1672501744` будет иметь значение `15` и изменить его уже нельзя. Если тип данных изменяемый, то можно менять значение объекта.

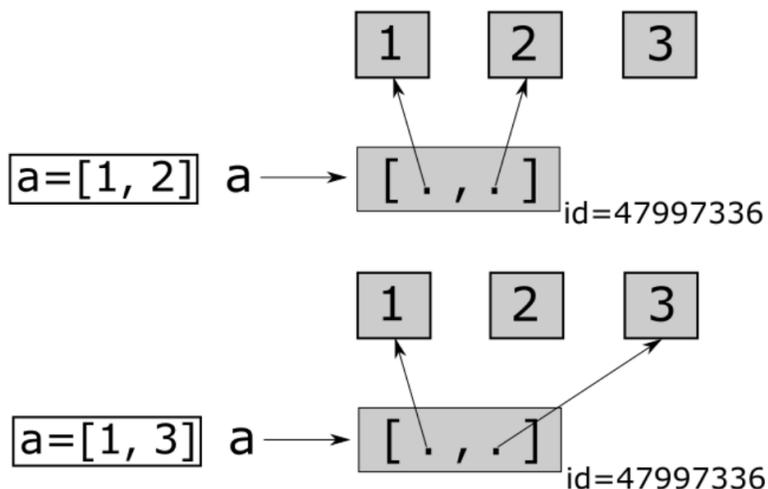
Например, создадим список `[1, 2]`, а потом заменим второй элемент на `3`.

```
>>> a = [1, 2]
>>> id(a)
47997336
```

```
>>> a[1] = 3
>>> a
[ 1 , 3 ]
```

```
>>> id(a)
47997336
```

Как видно, объект на который ссылается переменная `a`, был изменен. Это можно проиллюстрировать следующим рисунком.



В рассмотренном случае, в качестве данных списка, выступают не объекты, а отношения между объектами. Т.е. в переменной `a` хранятся ссылки на объекты содержащие числа 1 и 3, а не непосредственно сами эти числа.

4. Целочисленные типы

В языке Python имеется два целочисленных типа, `int` и `bool`. И целые числа, и логические значения являются неизменяемыми объектами, но благодаря присутствию в языке Python комбинированных операторов присваивания эта особенность практически незаметна. В логических выражениях число `0` и значение `False` представляют `False`, а любое другое целое число и значение `True` представляют `True`. В числовых выражениях значение `True` представляет 1, а `False` – 0. Это означает, что можно записывать весьма странные выражения, например, выражение `i += True` увеличит значение `i` на единицу. Естественно, более правильным будет записывать подобные выражения как `i += 1`.

Размер целого числа ограничивается только объемом памяти компьютера, поэтому легко можно создать и обрабатывать целое число, состоящее из тысяч цифр, правда, скорость работы с такими числами существенно медленнее, чем с числами, которые соответствуют машинному представлению.

Литералы целых чисел по умолчанию записываются в десятичной системе счисления, но при желании можно использовать другие системы счисления:

```
>>> 14600926
14600926
>>> 0b110111101100101011011110
14600926
```

```
>>> 0o67545336
14600926
>>> 0xDECADE
14600926
```

Двоичные числа записываются с префиксом `0b`, восьмеричные – в префиксом `0o` и шестнадцатеричные – с префиксом `0x`. В префиксах допускается использовать символы верхнего регистра.

При работе с целыми числами могут использоваться обычные математические функции и операторы, как показано в табл. 1. Для арифметических операций `+`, `-`, `/`, `//`, `%` и `**` имеются соответствующие комбинированные операторы присваивания: `+=`, `-=`, `/=`, `//=`, `%=` и `**=`, где выражение `x op= y` является эквивалентом выражения `x = x op y`.

Таблица 1: Арифметические операторы и функции

Синтаксис	Описание
<code>x + y</code>	Складывает число <code>x</code> и число <code>y</code>
<code>x - y</code>	Вычитает число <code>y</code> из числа <code>x</code>
<code>x * y</code>	Умножает <code>x</code> на <code>y</code>
<code>x / y</code>	Делит <code>x</code> на <code>y</code> – результатом всегда является значение типа <code>float</code> (или <code>complex</code> , если <code>x</code> или <code>y</code> является комплексным числом)
<code>x // y</code>	Делит <code>x</code> на <code>y</code> , при этом отсекает дробную часть, поэтому результатом всегда является значение типа <code>int</code> ; смотрите также функцию <code>round()</code>
<code>x % y</code>	Возвращает модуль (остаток) от деления <code>x</code> на <code>y</code>
<code>x**y</code>	Возводит <code>x</code> в степень <code>y</code> ; смотрите также функцию <code>pow()</code>
<code>-x</code>	Изменяет знак числа <code>x</code> , если оно не является нулем, если ноль – ничего не происходит
<code>+x</code>	Ничего не делает иногда используется для повышения удобочитаемости программного кода
<code>abs(x)</code>	Возвращает абсолютное значение <code>x</code>
<code>divmod(x, y)</code>	Возвращает частное и остаток деления <code>x</code> на <code>y</code> в виде кортежа двух значений типа <code>int</code>
<code>pow(x, y)</code>	Возводит <code>x</code> в степень <code>y</code> ; то же самое что и оператор <code>**</code>
<code>pow(x, y, z)</code>	Более быстрая альтернатива выражению <code>(x ** y) % z</code>
<code>round(x, n)</code>	Возвращает значение типа <code>int</code> , соответствующее значению <code>x</code> типа <code>float</code> , округленному до ближайшего целого числа (или значение типа <code>float</code> , округленное до <code>n</code> -го знака после запятой, если задан аргумент <code>n</code>)

Объекты могут создаваться путем присваивания литералов переменным, например, `x = 17`, или обращением к имени соответствующего типа как к функции, например, `x = int(17)`. Создание объекта посредством использования его типа может быть выполнено одним из трех способов:

- вызов типа данных без аргументов. В этом случае объект приобретает значение по умолчанию, например, выражение `x = int()` создаст целое число `0`. Любые встроенные типы могут вызываться без аргументов.
- тип вызывается с единственным аргументом. Если указан аргумент соответствующего типа, будет создана поверхностная копия оригинального объекта. Если задан аргумент другого типа, будет предпринята попытка выполнить преобразование. Такой способ использования описывается в табл. 2
- передается два или более аргументов; не все типы поддерживают такую возможность, а для тех типов, что поддерживают ее, типы аргументов и их назначение отличаются. В случае типа `int` допускается передавать два аргумента, где первый аргумент – это строка с представлением целого числа, а второй аргумент – число основания системы числения. Например, вызов `int("A4", 16)` создаст десятичное значение `164`.

Таблица 2: Функции преобразования целых чисел

Синтаксис	Описание
<code>bin(i)</code>	Возвращает двоичное представление целого числа <code>i</code> в виде строки, например, <code>bin(1980) == '0b11110111100'</code>
<code>hex(i)</code>	Возвращает шестнадцатеричное представление целого числа <code>i</code> в виде строки, например, <code>hex(1980) == '0x7bc'</code>
<code>int(x)</code>	Преобразует объект <code>x</code> в целое число; в случае ошибки во время преобразования возбуждает исключение <code>ValueError</code> , а если тип объекта <code>x</code> не поддерживает преобразование в целое число возбуждает исключение <code>TypeError</code> . Если <code>x</code> является числом с плавающей точкой, оно преобразуется в целое число путем усечения дробной части.
<code>int(s, base)</code>	Преобразует строку <code>s</code> в целое число; в случае ошибки возбуждает исключение <code>ValueError</code> . Если задан необязательный аргумент <code>base</code> , он должен быть целым числом в диапазоне от 2 до 36 включительно.
<code>oct(i)</code>	Возвращает восьмеричное представление целого числа <code>i</code> в виде строки, например, <code>oct(1980) == '0o3674'</code>

В табл. 3 перечислены битовые операторы. Все битовые операторы (`|`, `^`, `&`, « и ») имеют соответствующие комбинированные операторы присваивания (`|=`, `^=`, `&=`, « и »=`=`), где выражение `i op= j` является логическим эквивалентом выражения `i = i op j` в случае, когда обращение к значению `i` не имеет побочных эффектов.

Таблица 3: Функции преобразования целых чисел

Синтаксис	Описание
<code>i j</code>	Битовая операция OR (ИЛИ) над целыми числами <code>i</code> и <code>j</code> ; отрицательные числа представляются как двоичное дополнение
<code>i ^ j</code>	Битовая операция XOR (исключающее ИЛИ) над целыми числами <code>i</code> и <code>j</code>
<code>i & j</code>	Битовая операция AND (И) над целыми числами <code>i</code> и <code>j</code>
<code>i « j</code>	Сдвигает значение <code>i</code> влево на <code>j</code> битов аналогично операции <code>i * (2 ** j)</code> без проверки на переполнение
<code>i » j</code>	Сдвигает значение <code>i</code> вправо на <code>j</code> битов аналогично операции <code>i // (2 ** j)</code> без проверки на переполнение
<code>~i</code>	Инвертирует биты числа <code>i</code>

4.1. Логические значения

Существует два встроенных логических объекта: `True` и `False`. Как и все остальные типы данных в языке Python (встроенные, библиотечные или ваши собственные), тип данных `bool` может вызываться как функция – при вызове без аргументов возвращается значение `False`, при вызове с аргументом типа `bool` возвращается копия аргумента, а при вызове с любым другим аргументом предпринимается попытка преобразовать указанный объект в тип `bool`. Все встроенные типы данных и типы данных из стандартной библиотеки могут быть преобразованы в тип `bool`, а добавить поддержку такого преобразования в свои собственные типы данных не представляет никакой сложности. Ниже приводится пара присваиваний логических значений и пара логических выражений:

```
>>> t = True
>>> f = False
>>> t and f
False
>>> t and True
True
```

в языке Python имеется три логических оператора: `and`, `or` и `not`. Выражения с участием операторов `and` и `or` вычисляются в соответствии с логикой сокращенных вычислений (*short-circuit logic*), и возвращается операнд, определяющий значение всего выражения, тогда как результатом оператора `not` всегда является либо `True`, либо `False`.

5. Типы чисел с плавающей точкой

Язык Python предоставляет три типа значений с плавающей точкой: встроенные типы `float` и `complex` и тип `decimal.Decimal` в стандартной библиотеке. Все три типа данных относятся к категории неизменяемых. Тип `float` представляет числа с плавающей точкой двойной точности, диапазон значений которых зависит от компилятора языка C (или C# или Java), применявшегося для компиляции интерпретатора Python. Числа этого типа имеют ограниченную точность и не могут надежно сравниваться на равенство значений. Числа типа `float`

записываются с десятичной точкой или в экспоненциальной форме записи, например, `0.0`, `4.`, `5.7`, `-2.5`, `-2e9`, `8.9e-4`.

В машинном представлении числа с плавающей точкой хранятся как двоичные числа. Это означает, что одни дробные значения могут быть представлены точно (такие как `0.5`), а другие – только приблизительно (такие как `0.1` и `0.2`). Кроме того, для представления используется фиксированное число битов, поэтому существует ограничение на количество цифр в представлении таких чисел. Ниже приводится поясняющий пример:

```
>>> 0.0, 5.4, -2.5, 8.9e-4
```

Проблема потери точности – это не проблема, свойственная только языку Python; все языки программирования обнаруживают проблему с точным представлением чисел с плавающей точкой.

Если действительно необходимо обеспечить высокую точность, можно использовать числа типа `decimal.Decimal`. Эти числа обеспечивают уровень точности, который вы укажете (по умолчанию 28 знаков после запятой), и могут точно представлять периодические числа, такие как `0.1`, но скорость работы с такими числами существенно ниже, чем с обычными числами типа `float`. Вследствие высокой точности числа типа `decimal.Decimal` прекрасно подходят для производства финансовых вычислений.

Смешанная арифметика поддерживается таким образом, что результатом выражения с участием чисел типов `int` и `float` является число типа `float`, а с участием типов `float` и `complex` результатом является число типа `complex`. Поскольку числа типа `decimal.Decimal` имеют фиксированную точность, они могут участвовать в выражениях только с другими числами `decimal.Decimal` и с числами типа `int`; результатом таких выражений является число `decimal.Decimal`. В случае попытки выполнить операцию над несовместимыми типами возбуждается исключение `TypeError`.

5.1. Числа с плавающей точкой

Все числовые операторы и функции, представленные в табл. 1, могут применяться к числам типа `float`, включая комбинированные операторы присваивания. Тип данных `float` может вызываться как функция – без аргументов возвращается число `0.0`, с аргументом типа `float` возвращается копия аргумента, а с аргументом любого другого типа предпринимается попытка выполнить преобразование указанного объекта в тип `float`. При преобразовании строки аргумент может содержать либо простую форму записи числа с десятичной точкой, либо экспоненциальное представление числа. При выполнении операций с числами типа `float` может возникнуть ситуация, когда в результате получается значение `NaN` (*not a number* – не число) или «бесконечность». К сожалению, поведение интерпретатора в таких ситуациях может отличаться в разных реализациях и зависит от математической библиотеки системы.

Ниже приводится пример простой функции, выполняющей сравнение чисел типа `float` на равенство в пределах машинной точности:

```
def equal_float(a, b):  
    return abs(a - b) <= sys.float_info.epsilon
```

Чтобы воспользоваться этой функцией, необходимо импортировать модуль `sys`. Объект `sys.float_info` имеет множество атрибутов. Так, `sys.float_info.epsilon` хранит минимально возможную разницу между двумя числами с плавающей точкой. На одной из 32-разрядных машин автора книги это число чуть больше 0.0000000000000002. Тип `float` в языке Python обеспечивает надежную точность до 17 значащих цифр.

В дополнение к встроенным функциональным возможностям работы с числами типа `float` модуль `math` предоставляет множество функций, которые приводятся в табл. 4. Ниже приводятся несколько фрагментов программного кода, демонстрирующих, как можно использовать функциональные возможности модуля:

```
>>> import math
>>> math.pi * (5 ** 2)
78.539816339744831
>>> math.hypot(5, 12)
13.0
>>> math.modf(13.732)
(0.7319999999999932, 13.0)
```

Модуль `math` в значительной степени опирается на математическую библиотеку, с которой был собран интерпретатор Python. Это означает, что при некоторых условиях и в граничных случаях функции модуля могут иметь различное поведение на различных платформах.

Таблица 4: Функции и константы модуля `math`

Синтаксис	Описание
<code>math.acos(x)</code>	Возвращает арккосинус x в радианах
<code>math.acosh(x)</code>	Возвращает гиперболический арккосинус x в радианах
<code>math.asin(x)</code>	Возвращает арксинус x в радианах
<code>math.asinh(x)</code>	Возвращает гиперболический арксинус x в радианах
<code>math.atan(x)</code>	Возвращает арктангенс x в радианах
<code>math.atan2(y, x)</code>	Возвращает арктангенс y/x в радианах
<code>math.atanh(x)</code>	Возвращает гиперболический арктангенс x в радианах
<code>math.ceil(x)</code>	Возвращает $ x $, то есть наименьшее целое число типа <code>int</code> , большее и равное x , например, <code>math.ceil(5.4) == 6</code>
<code>math.copysign(x, y)</code>	Возвращает x со знаком числа y
<code>math.cos(x)</code>	Возвращает косинус x в радианах
<code>math.cosh(x)</code>	Возвращает гиперболический косинус x в радианах
<code>math.degrees(r)</code>	Преобразует число r типа <code>float</code> из радианов в градусы
<code>math.e</code>	Константа e , примерно равная значению 2.7182818284590451
<code>math.exp(x)</code>	Возвращает e^x , то есть <code>math.e ** x</code>
<code>math.fabs(x)</code>	Возвращает $ x $, то есть абсолютное значение x в виде числа типа <code>float</code>
<code>math.factorial(x)</code>	Возвращает $x!$

<code>math.floor(x)</code>	Возвращает $ x $, то есть наименьшее целое число типа <code>int</code> , меньшее и равное <code>x</code> , например, <code>math.floor(5.4) == 5</code>
<code>math.fmod(x y)</code>	Выполняет деление по модулю (возвращает остаток) числа <code>x</code> на число <code>y</code> ; дает более точный результат, чем оператор <code>%</code> , применительно к числам типа <code>float</code>
<code>math.frexp(x)</code>	Возвращает кортеж из двух элементов с мантиссой (в виде числа типа <code>float</code>) и экспонентой (в виде числа типа <code>int</code>)
<code>math.fsum(i)</code>	Возвращает сумму значений в итерируемом объекте <code>i</code> в виде числа типа <code>float</code>
<code>math.hypot(x y)</code>	Возвращает $\sqrt{x^2 + y^2}$
<code>math.isinf(x)</code>	Возвращает <code>True</code> , если значение <code>x</code> типа <code>float</code> является бесконечностью ($\pm\infty$)
<code>math.isnan(x)</code>	Возвращает <code>True</code> , если значение <code>x</code> типа <code>float</code> не является числом
<code>math.ldexp(m e)</code>	Возвращает $m \times 2^e$ – операция обратная <code>math.frexp()</code>
<code>math.log(x b)</code>	Возвращает $\log_b x$, аргумент <code>b</code> является необязательным и по умолчанию имеет значение <code>math.e</code>
<code>math.log10(x)</code>	Возвращает $\log_{10} x$
<code>math.log1p(x)</code>	Возвращает $\log_e(1+x)$; дает точные значения даже когда значение <code>x</code> близко к <code>0</code>
<code>math.modf(x)</code>	Возвращает дробную и целую часть числа <code>x</code> в виде двух значений типа <code>float</code>
<code>math.pi</code>	Константа π , примерно равная 3.1415926535897931
<code>math.pow(x y)</code>	Возвращает x^y в виде числа типа <code>float</code>
<code>math.radians(d)</code>	Преобразует число <code>d</code> типа <code>float</code> из градусов в радианы
<code>math.sin(x)</code>	Возвращает синус <code>x</code> в радианах
<code>math.sinh(x)</code>	Возвращает гиперболический синус <code>x</code> в радианах
<code>math.sqrt(x)</code>	Возвращает \sqrt{x}
<code>math.tan(x)</code>	Возвращает тангенс <code>x</code> в радианах
<code>math.tanh(x)</code>	Возвращает гиперболический тангенс <code>x</code> в радианах
<code>math.trunc(x)</code>	Возвращает целую часть числа <code>x</code> в виде значения типа <code>int</code> ; то же самое что и <code>int(x)</code>

5.2. Комплексные числа

Тип данных `complex` относится к категории неизменяемых и хранит пару значений типа `float`, одно из которых представляет действительную часть комплексного числа, а другое – мнимую. Литералы комплексных чисел записываются как действительная и мнимая части, объединенные знаком `+` или `-`, а за мнимой частью числа следует символ `j`. Вот примеры нескольких комплексных чисел: `3.5+2j`, `0.5j`, `4+0j`, `-1 - 3.7j`. Обратите внимание, что если действительная часть числа равна `0`, ее можно вообще опустить.

Отдельные части комплексного числа доступны в виде атрибутов `real` и `imag`. Например:

```
>>> z = -89.5+2.125j
>>> z.real, z.imag
(-89.5, 2.125)
```

За исключением `//`, `%`, `divmod()` и версии `pow()` с тремя аргументами все остальные арифметические операторы и функции, перечисленные в табл. 1 могут использоваться для работы с комплексными числами, так же как и соответствующие комбинированные операторы присваивания. Кроме того, значения типа `complex` имеют метод `conjugate()`, который изменяет знак мнимой части. Например:

```
>>> z.conjugate()
(-89.5-2.125j)
```

```
>>> 3-4j.conjugate()
(3+4j)
```

Тип данных `complex` может вызываться как функция – без аргументов она вернет значение `0j`, с аргументом типа `complex` она вернет копию аргумента, а с аргументом любого другого типа она попытается преобразовать указанный объект в значение типа `complex`. При использовании для преобразования функция `complex()` принимает либо единственный строковый аргумент, либо одно или два значения типа `float`.

Если ей передается единственное значение типа `float`, возвращается комплексное число с мнимой частью, равной `0j`.

Функции в модуле `math` не работают с комплексными числами. Это сделано преднамеренно, чтобы гарантировать, что пользователи модуля `math` будут получать исключения вместо получения комплексных чисел в некоторых случаях.

Если возникает необходимость использовать комплексные числа, можно воспользоваться модулем `cmath`, который содержит комплексные версии большинства тригонометрических и логарифмических функций, присутствующих в модуле `math`, плюс ряд функций, специально предназначенных для работы с комплексными числами, таких как `cmath.phase()`, `cmath.polar()` и `cmath.rect()`, а также константы `cmath.pi` и `cmath.e`, которые хранят те же самые значения типа `float`, что и родственные им константы в модуле `math`.

5.3. Числа типа `Decimal`

Во многих приложениях недостаток точности, свойственный числам типа `float`, не имеет существенного значения, и эта неточность окупается скоростью вычислений. Но в некоторых случаях предпочтение отдается точности, даже в обмен на снижение скорости работы. Модуль `decimal` реализует неизменяемый числовой тип `Decimal`, который представляет числа с задаваемой точностью. Вычисления с участием таких чисел производятся значительно медленнее, чем в случае использования значений типа `float`, но насколько это важно, будет зависеть от приложения.

Чтобы создать объект типа `Decimal`, необходимо импортировать модуль `decimal`. Например:

```
>>> import decimal
>>> a = decimal.Decimal(9876)
>>> b = decimal.Decimal("54321.012345678987654321")
>>> a + b
Decimal('64197.012345678987654321')
```

6. Строки

Строки в языке Python представлены неизменяемым типом данных `str`, который хранит последовательность символов Юникода. Тип данных `str` может вызываться как функция для создания строковых объектов – без аргументов возвращается пустая строка; с аргументом, который не является строкой, возвращается строковое представление аргумента; а в случае, когда аргумент является строкой, возвращается его копия. Функция `str()` может также использоваться как функция преобразования. В этом случае первый аргумент должен быть строкой или объектом, который можно преобразовать в строку, а, кроме того, функции может быть передано до двух необязательных строковых аргументов, один из которых определяет используемую кодировку, а второй определяет порядок обработки ошибок кодирования.

Литералы строк создаются с использованием кавычек или апострофов, при этом важно, чтобы с обоих концов литерала использовались кавычки одного и того же типа. В дополнение к этому мы можем использовать строки в тройных кавычках, то есть строки, которые начинаются и заканчиваются тремя символами кавычки (либо тремя кавычками, либо тремя апострофами). Например:

```
text = """Строки в тройных кавычках могут включать 'апострофы' и "кавычки"
без лишних формальностей. Мы можем даже экранировать символ перевода строки \,
благодаря чему данная конкретная строка будет занимать всего две строки."""
```

Если нам потребуется использовать кавычки в строке, это можно сделать без лишних формальностей – при условии, что они отличаются от кавычек, ограничивающих строку; в противном случае символы кавычек или апострофов внутри строки следует экранировать:

```
a = "Здесь 'апострофы' можно не экранировать, а \"кавычки\" придется."
b = 'Здесь \"апострофы\" придется экранировать, а \"кавычки\" не обязательно.'
```

В языке Python символ перевода строки интерпретируется как завершающий символ инструкции, но не внутри круглых скобок (`()`), квадратных скобок (`[]`), фигурных скобок (`{}`) и строк в тройных кавычках. Символы перевода строки могут без лишних формальностей использоваться в строках в тройных кавычках, и мы можем включать символы перевода строки в любые строковые литералы с помощью экранированной последовательности `\n`.

Все экранированные последовательности, допустимые в языке Python, перечислены в табл. 2.6.

Таблица 5: Функции и константы модуля `math`

Последовательность	Значение
<code>\переводстроки</code>	Экранирует (то есть игнорирует) символ перевода строки
<code>\\</code>	Символ обратного следа (<code>\</code>)
<code>\'</code>	Апостроф (<code>'</code>)
<code>\"</code>	Кавычка (<code>"</code>)
<code>\a</code>	Символ ASCII «сигнал» (bell, BEL)
<code>\b</code>	Символ ASCII «забой» (backspace, BS)
<code>\f</code>	Символ ASCII «перевод формата» (formfeed, FF)
<code>\n</code>	Символ ASCII «перевод строки» (linefeed, LF)
<code>\N{название}</code>	Символ Юникода с заданным названием
<code>\ooo</code>	Символ с заданным восьмеричным кодом
<code>\r</code>	Символ ASCII «возврат каретки» (carriage return, CR)
<code>\t</code>	Символ ASCII «табуляция» (tab, TAB)
<code>\uhhhh</code>	Символ Юникода с указанным 16-битовым шестнадцатеричным значением
<code>\Uhhhhhhhh</code>	Символ Юникода с указанным 32-битовым шестнадцатеричным значением
<code>\v</code>	Символ ASCII «вертикальная табуляция» (vertical tab, VT)
<code>\xhh</code>	Символ с указанным 8-битовым шестнадцатеричным значением

Если потребуется записать длинный строковый литерал, занимающий две или более строк, но без использования тройных кавычек, то можно использовать один из приемов, показанных ниже:

```
t = "Это не самый лучший способ объединения двух длинных строк, " + \
    "потому что он основан на использовании неуклюжего экранирования"
s = ("Это отличный способ объединить две длинные строки, "
    " потому что он основан на конкатенации строковых литералов.")
```

Обратите внимание, что во втором случае для создания единственного выражения мы должны были использовать круглые скобки – без этих скобок переменной `s` была бы присвоена только первая строка, а наличие второй строки вызвало бы исключение `IndentationError`.

6.1. Сравнение строк

Строки поддерживают обычные операторы сравнения `<`, `<=`, `=`, `!=`, `>` и `>=`. Эти операторы выполняют побайтовое сравнение строк в памяти. К сожалению, возникают две проблемы при сравнении, например, строк в отсортированных списках. Обе проблемы проявляются во всех языках программирования и не являются характерной особенностью Python.

Первая проблема связана с тем, что символы Юникода могут быть представлены двумя и более последовательностями байтов.

Вторая проблема заключается в том, что порядок сортировки некоторых символов зависит от конкретного языка.

6.2. Получение срезов строк

Отдельные элементы последовательности, а, следовательно, и отдельные символы в строках, могут извлекаться с помощью оператора доступа к элементам (`[]`). В действительности этот оператор намного более универсальный и может использоваться для извлечения не только одного символа, но и целых комбинаций (подпоследовательностей) элементов или символов, когда этот оператор используется в контексте оператора извлечения среза.

Для начала мы рассмотрим возможность извлечения отдельных символов. Нумерация позиций символов в строках начинается с 0 и продолжается до значений длины строки минус 1. Однако допускается использовать и отрицательные индексы – в этом случае отсчет начинается с последнего символа и ведется в обратном направлении к первому символу. На рис. 1 показано, как нумеруются позиции символов в строке, если предположить, что было выполнено присваивание `s = "Light ray"`.

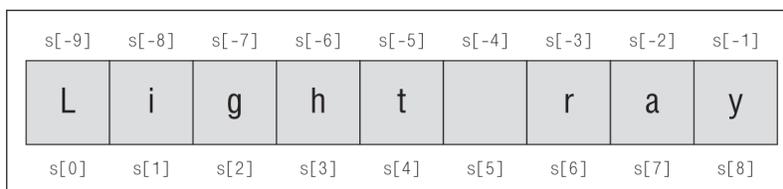


Рис. 1: Номера позиций символов в строке

Отрицательные индексы удивительно удобны, особенно индекс `-1`, который всегда соответствует последнему символу строки. Попытка обращения к индексу, находящемуся за пределами строки (или к любому индексу в пустой строке), будет вызывать исключение `IndexError`. Оператор получения среза имеет три формы записи:

```
seq[start]
seq[start:end]
seq[start:end:step]
```

Ссылка `seq` может представлять любую последовательность, такую как список, строку или кортеж. Значения `start`, `end` и `step` должны быть целыми числами (или переменными, хранящими целые числа). Первая форма — это запись оператора доступа к элементам: с ее помощью извлекается элемент последовательности с индексом `start`. Вторая форма записи извлекает подстроку, начиная с элемента с индексом `start` и заканчивая элементом с индексом `end`, не включая его.

При использовании второй формы записи (с одним двоеточием) мы можем опустить любой из индексов. Если опустить начальный индекс, по умолчанию будет использоваться значение `0`. Если опустить конечный индекс, по умолчанию будет использоваться значение

`len(seq)`. Это означает, что если опустить оба индекса, например, `s[:]`, это будет равносильно выражению `s[0:len(s)]`, и в результате будет извлечена, то есть скопирована, последовательность целиком.

На рис. 2 приводятся некоторые примеры извлечения срезов из строки `s`, которая получена в результате присваивания `s = "The waxwork man"`.

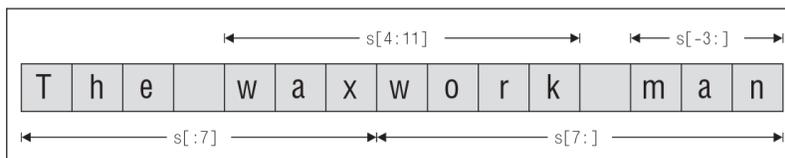


Рис. 2: Извлечение срезов из последовательности

Один из способов вставить подстроку в строку состоит в смешивании операторов извлечения среза и операторов конкатенации. Например:

```
>>> s = s[:12] + "wo" + s[12:]
>>> s
'The waxwork woman'
```

Кроме того, поскольку текст «wo» присутствует в оригинальной строке, тот же самый эффект можно было бы получить путем присваивания значения выражения `s[:12] + s[7:9] + s[12:]`.

Оператор конкатенации `+` и добавления подстроки `+=` не особенно эффективны, когда в операции участвует множество строк. Для объединения большого числа строк обычно лучше использовать метод `str.join()`, с которым мы познакомимся в следующем подразделе.

Третья форма записи (с двумя двоеточиями) напоминает вторую форму, но в отличие от нее значение `step` определяет, с каким шагом следует извлекать символы. Как и при использовании второй формы записи, мы можем опустить любой из индексов. Если опустить начальный индекс, по умолчанию будет использоваться значение `0`, при условии, что задано неотрицательное значение `step`; в противном случае начальный индекс по умолчанию получит значение `-1`. Если опустить конечный индекс, по умолчанию будет использоваться значение `len(seq)`, при условии, что задано неотрицательное значение `step`; в противном случае конечный индекс по умолчанию получит значение индекса перед началом строки. Мы не можем опустить значение `step`, и оно не может быть равно нулю – если задание шага не требуется, то следует использовать вторую форму записи (с одним двоеточием), в которой шаг выбора элементов не указывается.

На рис. 3 приводятся пара примеров извлечения разреженных срезов из строки `s`, которая получена в результате присваивания `s = "he ate camel food"`.

Здесь мы использовали значения по умолчанию для начального и конечного индексов, то есть извлечение среза `s[::-2]` начинается с последнего символа строки и извлекается каждый второй символ по направлению к началу строки. Аналогично извлечение среза `s[::3]` начинается с первого символа строки и извлекается каждый третий символ по направлению к концу строки. Существует возможность комбинировать индексы с размером шага, как показано на рис. 4.

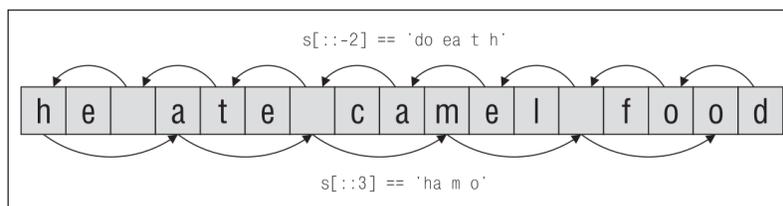


Рис. 3: Извлечение разреженных срезов

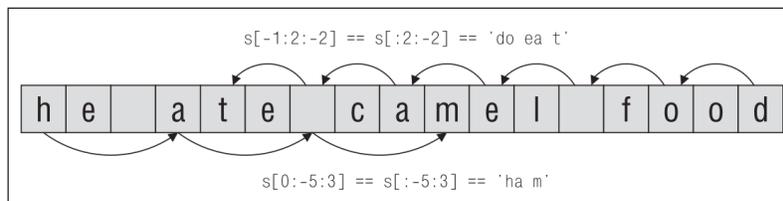


Рис. 4: Извлечение срезов из последовательности с определенным шагом

Операция извлечения элементов с определенным шагом часто применяется к последовательностям, отличным от строк, но один из ее вариантов часто применяется к строкам:

6.3. Операторы и методы строк

Поскольку строки относятся к категории неизменяемых последовательностей, все функциональные возможности, применимые к неизменяемым последовательностям, могут использоваться и со строками. Сюда входят оператор проверки на вхождение `in`, оператор конкатенации `+`, оператор добавления в конец `+=`, оператор дублирования `*` и комбинированный оператор присваивания с дублированием `*=`. Применение всех этих операторов в контексте строк мы обсудим в этом подразделе, а также обсудим большинство строковых методов. В табл. 2.7 приводится перечень некоторых строковых методов.

Так как строки являются последовательностями, они являются объектами, имеющими «размер», и поэтому мы можем вызывать функцию `len()`, передавая ей строки в качестве аргумента. Возвращаемая функцией длина представляет собой количество символов в строке (ноль – для пустых строк).

Мы уже знаем, что перегруженная версия оператора `+` для строк выполняет операцию конкатенации. В случаях, когда требуется объединить множество строк, лучше использовать метод `str.join()`. Метод принимает в качестве аргумента последовательность (то есть список или кортеж строк) и объединяет их в единую строку, вставляя между ними строку, относительно которой был вызван метод. Например:

```
>>> treatises = ["Arithmetica", "Conics", "Elements"]
>>> " ".join(treatises)
'Arithmetica Conics Elements'
```

```
>>> "-<>".join(treatises)
'Arithmetica-<>-Conics-<>-Elements'
```

```
>>> "".join(treatises)
'ArithmeticaConicsElements'
```

Метод `str.join()` может также использоваться в комбинации со встроенной функцией `reversed()`, которая переворачивает строку – например, `"".join(reversed(s))`, хотя тот же результат может быть получен более кратким оператором извлечения разреженного среза – например, `s[::-1]`.

Оператор `*` обеспечивает возможность дублирования строки:

```
>>> s = "_" * 5
>>> print(s)
=====
```

```
>>> s *= 10
>>> print(s)
=====
```

Как показано в примере, мы можем также использовать комбинированный оператор присваивания с дублированием.

6.4. Форматирование строк с помощью метода `str.format()`

Метод `str.format()` представляет собой очень мощное и гибкое средство создания строк. Использование метода `str.format()` в простых случаях не вызывает сложностей, но для более сложного форматирования нам необходимо изучить синтаксис форматирования.

Метод `str.format()` возвращает новую строку, замещая поля в контекстной строке соответствующими аргументами. Например:

```
>>> "The novel '{0}' was published in {1}".format("Hard Times", 1854)
"The novel 'Hard Times' was published in 1854"
```

Каждое замещаемое поле идентифицируется именем поля в фигурных скобках. Если в качестве имени поля используется целое число, оно определяет порядковый номер аргумента, переданного методу `str.format()`. Поэтому в данном случае поле с именем `0` было замещено первым аргументом, а поле с именем `1` – вторым аргументом.

Если бы нам потребовалось включить фигурные скобки в строку формата, мы могли бы сделать это, дублируя их, как показано ниже:

```
>>> "{{{0}}} {1} ;-)".format("I'm in braces", "I'm not")
"{I'm in braces} I'm not ;-)"
```

Если попытаться объединить строку и число, интерпретатор Python совершенно справедливо возбудит исключение `TypeError`. Но это легко можно сделать с помощью метода `str.format()`:

```
>>> "{0}{1}".format("The amount due is $", 200)
'The amount due is $200'
```

С помощью `str.format()` мы также легко можем объединять строки (хотя для этой цели лучше подходит метод `str.join()`):

```
>>> x = "three"
>>> s = "{0} {1} {2}"
>>> s = s.format("The", x, "tops")
>>> s
'The three tops'
```

В следующем разделе мы рассмотрим применение функции `str.format()`.

7. Примеры

7.1. Печать символов Юникода

Рассмотрим небольшой, но достаточно поучительный пример использования метода `str.format()`, в котором мы увидим применение спецификаторов формата в реальном контексте. Программа, состоящая всего из 25 строк выполняемого кода, находится в файле `print_unicode.py`. Она импортирует два модуля, `sys` и `unicodedata` и определяет одну функцию – `print_unicode_table()`. Рассмотрение примера мы начнем с запуска программы, чтобы увидеть, что она делает; затем мы рассмотрим программный код в конце программы, где выполняется вся фактическая работа; и в заключение рассмотрим функцию, определяемую в программе.

```
Terminal
> python print_unicode.py "circled number"
decimal hex chr name
-----
9321 2469 ⑩ Circléd Number Ten
9322 246A ⑪ Circléd Number Eleven
9323 246B ⑫ Circléd Number Twelve
9324 246C ⑬ Circléd Number Thirteén
9325 246D ⑭ Circléd Number Fourteen
9326 246E ⑮ Circléd Number Fifteen
9327 246F ⑯ Circléd Number Sixteen
9328 2470 ⑰ Circléd Number Seventeen
9329 2471 ⑱ Circléd Number Eighteen
9330 2472 ⑲ Circléd Number Nineteén
9331 2473 ⑳ Circléd Number Twenty
9451 24EB ㊱ Negative Circléd Number Eleven
9452 24EC ㊲ Negative Circléd Number Twelve
9453 24ED ㊳ Negative Circléd Number Thirteén
9454 24EE ㊴ Negative Circléd Number Fourteen
9455 24EF ㊵ Negative Circléd Number Fifteen
```

9456	24F0	Ⓜ	Negative Circled Number Sixteen
9457	24F1	Ⓜ	Negative Circled Number Seventeen
9458	24F2	Ⓜ	Negative Circled Number Eighteen
9459	24F3	Ⓜ	Negative Circled Number Nineteen
9460	24F4	Ⓜ	Negative Circled Number Twenty

При запуске без аргументов программа выводит таблицу всех символов Юникода, начиная с пробела и до символа с наибольшим возможным кодом. При запуске с аргументом, как показано в примере, выводятся только те строки таблицы, где в названии символов Юникода содержится значение строки-аргумента, переведенной в нижний регистр.

Разберем исходный код программы:

```
# Start main script
word = None

if len(sys.argv) > 1:
    if sys.argv[1] in ("-h", "--help"):
        print("usage: {0} [string]".format(sys.argv[0]))
        word = 0
    else:
        word = sys.argv[1].lower()

if word != 0:
    print_unicode_table(word)
```

После инструкций импортирования и определения функции `print_unicode_table()` выполнение достигает программного кода, показанного выше. Сначала предположим, что пользователь не указал в командной строке искомое слово. Если аргумент командной строки присутствует и это `-h` или `-help`, программа выводит информацию о порядке использования и устанавливает флаг `word` в значение `0`, указывая тем самым, что работа завершена. В противном случае в переменную `word` записывается копия аргумента, введенного пользователем, с преобразованием всех символов в нижний регистр. Если значение `word` не равно `0`, программа выводит таблицу.

При выводе информации о порядке использования применяется спецификатор формата, который представляет собой простое имя формата, в данном случае – порядковый номер позиционного аргумента. Мы могли бы записать эту строку, как показано ниже:

```
print("usage: {0[0]} [string]".format(sys.argv))
```

При таком подходе первый символ `0` соответствует порядковому номеру позиционного аргумента, а `[0]` — это индекс элемента внутри аргумента, и такой прием работает, потому что `sys.argv` является списком.

```
def print_unicode_table(word):
    print("decimal hex chr {0:^40}".format("name"))
    print("----- ----- --- {0:~<40}".format(""))
```

```

code = ord(" ")
end = sys.maxunicode

while code < end:
    c = chr(code)
    name = unicodedata.name(c, "*** unknown ***")
    if word is None or word in name.lower():
        print("{0:7} {0:5X} {0:^3c} {1}".format(code, name.title()))
    code += 1

```

Первый вызов `str.format()` выводит текст "name", отцентрированный в поле вывода, шириной 40 символов, а второй вызов выводит пустую строку в поле шириной 40 символов, используя символ - в качестве символа-заполнителя, с выравниванием по левому краю.

Как вариант, вторую строку функции можно было записать, как показано ниже:

```

print("------ --- {0}".format("-" * 40))

```

Здесь мы использовали оператор дублирования строки (*), чтобы создать необходимую строку, и просто вставили ее в строку формата.

Текущий код символа Юникода сохраняется в переменной `code`, которая инициализируется кодом пробела (0x20). В переменную `end` записывается максимально возможный код символа Юникода, который может принимать значения в зависимости от того, какая из кодировок использовалась при компиляции Python.

Внутри цикла `while` с помощью функции `chr()` мы получаем символ Юникода, соответствующий числовому коду. Функция `unicodedata.name()` возвращает название заданного символа Юникода, во втором необязательном аргументе передается имя, которое будет использовано в случае, когда имя символа не определено.

Если пользователь не указывает аргумент командной строки (`word is None`) или аргумент был указан и он входит в состав копии имени символа Юникода, в которой все символы приведены к нижнему регистру, то выводится соответствующая строка таблицы.

Мы передаем переменную `code` методу `str.format()` один раз, но в строке формата она используется трижды. Первый раз – при выводе значения `code` как целого числа в поле с шириной 7 символов (по умолчанию в качестве символа-заполнителя используется пробел, поэтому нет необходимости явно указывать его). Второй раз – при выводе значения `code` как целого числа в шестнадцатеричном формате символами верхнего регистра в поле шириной 5 символов. И третий раз – при выводе символа Юникода, соответствующего значению `code`, с помощью спецификатора формата `c`, отцентрированного в поле с минимальной шириной 3 символа. Обратите внимание, что нам не потребовалось указывать тип `d` в первом спецификаторе формата, потому что он подразумевается по умолчанию для целых чисел. Второй аргумент – это имя символа Юникода, которое выводится с помощью метода `str.title()`, в результате которого первый символ каждого слова преобразуется к верхнему регистру, а остальные символы – к нижнему.

7.2. Решение квадратного уравнения

Квадратные уравнения – это уравнения вида $ax^2 + bx + c = 0$, где $a \neq 0$, описывающие параболу. Корни таких уравнений находятся по формуле

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Часть формулы $b^2 - 4ac$ называется дискриминантом – если это положительная величина, уравнение имеет два действительных корня, если дискриминант равен нулю – уравнение имеет один действительный корень, и в случае отрицательного значения уравнение имеет два комплексных корня. Мы напишем программу, которая будет принимать от пользователя коэффициенты a , b и c (коэффициенты b и c могут быть равны нулю) и затем вычислять и выводить его корень или корни.

Для начала посмотрим, как работает программа:

```
Terminal
```

```
> quadratic.py
ax 2 + bx + c = 0
enter a: 2.5
enter b: 0
enter c: -7.25
2.5x 2 + 0.0x + -7.25 = 0 → x = 1.70293863659 or x = -1.70293863659
```

С коэффициентами 1.5, -3 и 6 программа выведет (некоторые цифры обрезаны):

```
Terminal
```

```
> 1.5x 2 + -3.0x + 6.0 = 0 → x = (1+1.7320508j) or x = (1-1.7320508j)
```

Теперь обратимся к программному коду, который начинается тремя инструкциями `import`:

```
import cmath
import math
import sys
```

Нам необходимы обе математические библиотеки для работы с числами типа `float` и `complex`, так как функции, вычисляющие квадратный корень из вещественных и комплексных чисел, отличаются. Модуль `sys` нам необходим, так как в нем определена константа `sys.float_info.epsilon`, которая потребуется нам для сравнения вещественных чисел со значением `0`.

Нам также необходима функция, которая будет получать от пользователя число с плавающей точкой:

```
def get_float(msg, allow_zero):
    x = None
    while x is None:
        try:
            x = float(input(msg))
            if not allow_zero and abs(x) < sys.float_info.epsilon:
```

```

        print("zero is not allowed")
        x = None

    except ValueError as err:
        print(err)
return x

```

Эта функция выполняет цикл, пока пользователь не введет допустимое число с плавающей точкой (например, 0.5, -9, 21, 4.92), и допускает ввод значения 0, только если аргумент `allow_zero` имеет значение `True`. Вслед за определением функции `get_float()` выполняется оставшаяся часть программного кода. Мы разделим его на три части и начнем со взаимодействия с пользователем:

```

print("ax\N{SUPERSCRIPT TWO} + bx + c = 0")
a = get_float("enter a: ", False)
b = get_float("enter b: ", False)
c = get_float("enter c: ", False)

```

Благодаря функции `get_float()` получить значения коэффициентов `a`, `b` и `c` оказалось очень просто. Второй аргумент функции сообщает, когда значение 0 является допустимым.

```

x1 = None
x2 = None
discriminant = (b ** 2) - (4 * a * c)
if discriminant == 0:
    x1 = -(b / (2 * a))
else:
    if discriminant > 0:
        root = math.sqrt(discriminant)
    else:
        # discriminant < 0
        root = cmath.sqrt(discriminant)
    x1 = (-b + root) / (2 * a)
    x2 = (-b - root) / (2 * a)

```

Программный код выглядит несколько иначе, чем формула, потому что мы начали вычисления с определения значения дискриминанта. Если дискриминант равен 0, мы знаем, что уравнение имеет единственное действительное решение и можно сразу же вычислить его. В противном случае мы вычисляем действительный или комплексный квадратный корень из дискриминанта и находим два корня уравнения.

```

equation = ("{0}x\N{SUPERSCRIPT TWO} + {1}x + {2} = 0"
           "\N{RIGHTWARDS ARROW} x = {3}").format(a, b, c, x1)
if x2 is not None:
    equation += " or x = {0}".format(x2)
print(equation)

```

Мы не использовали сколько-нибудь сложного форматирования, поскольку форматирование, используемое по умолчанию для чисел с плавающей точкой в языке Python, прекрасно подходит для этого примера, но мы использовали некоторые имена Юникода для вывода пары специальных символов.

7.3. Представление таблицы csv в HTML

Часто бывает необходимо представить данные в формате HTML. В этом подразделе мы разработаем программу, которая читает данные из файла в простом формате CSV (Comma Separated Value – значения, разделенные запятыми) и выводит таблицу HTML, содержащую эти данные. В составе Python присутствует мощный и сложный модуль для работы с форматом CSV и похожими на него – модуль `csv`, но здесь мы будем выполнять всю обработку вручную.

В формате CSV каждая запись располагается на одной строке, а поля внутри записи отделяются друг от друга запятыми. Каждое поле может быть либо строкой, либо числом. Строки должны окружаться апострофами или кавычками, а числа не должны окружаться кавычками, если они не содержат запяты. Внутри строк допускается присутствие запятых, и они не должны интерпретироваться как разделители полей. Мы будем исходить из предположения, что первая запись в файле содержит имена полей. На выходе будет воспроизводиться таблица в формате HTML с выравниванием текста по левому краю (по умолчанию для HTML) и с выравниванием чисел по правому краю, по одной строке на запись и по одной ячейке на поле.

Ниже приводится маленький фрагмент файла с данными:

```
"COUNTRY",2000,2001,2002,2003,2004
"ANTIGUA AND BARBUDA",0,0,0,0,0
"ARGENTINA",37,35,33,36,39
"BAHAMAS, THE",1,1,1,1,1
"BAHRAIN",5,6,6,6,6
```

Предположим, что данные находятся в файле `sample.csv` и выполнена команда

```
Terminal
> python csv2html.py < sample.csv > sample.html
```

тогда файл `sample.html` должен содержать примерно следующее:

```
<table border='1'>
<tr bgcolor='lightgreen'>
<td>"Country"</td>
<td align='right'>2000</td>
<td align='right'>2001</td>
<td align='right'>2002</td>
<td align='right'>2003</td>
<td align='right'>2004</td>
</tr>
<tr bgcolor='white'>
<td>"Antigua and Barbuda"</td>
<td align='right'>0</td>
<td align='right'>0</td>
<td align='right'>0</td>
<td align='right'>0</td>
<td align='right'>0</td>
</tr>
<tr bgcolor='lightyellow'>
```

```

<td>"Argentina"</td>
<td align='right'>37</td>
<td align='right'>35</td>
<td align='right'>33</td>
<td align='right'>36</td>
<td align='right'>39</td>
</tr>
<tr bgcolor='white'>
<td>"Bahamas, The"</td>
<td align='right'>1</td>
<td align='right'>1</td>
<td align='right'>1</td>
<td align='right'>1</td>
<td align='right'>1</td>
</tr>
<tr bgcolor='lightyellow'>
<td>"Bahrain"</td>
<td align='right'>5</td>
<td align='right'>6</td>
<td align='right'>6</td>
<td align='right'>6</td>
<td align='right'>6</td>
</tr>
</table>

```

На рис. показано, как выглядит полученная таблица в веб-браузере.

"Country"	2000	2001	2002	2003	2004
"Antigua and Barbuda"	0	0	0	0	0
"Argentina"	37	35	33	36	39
"Bahamas, The"	1	1	1	1	1
"Bahrain"	5	6	6	6	6

Рис. 5: Таблица, произведенная программой `csv2html.py`, в браузере

Теперь, когда мы увидели, как используется программа и что она делает, можно приступить к изучению программного кода.

Последняя инструкция в программе – это простой вызов функции:

```
main()
```

Хотя в языке Python не требуется явно указывать точку входа в программу, как в некоторых других языках программирования, тем не менее является распространенной практикой создание в программе на языке Python функции с именем `main()`, которая вызывается для выполнения обработки. Поскольку функция не может вызываться до того, как она будет определена, мы должны вставлять вызов `main()` только после того, как данная функция

будет определена. Порядок следования функций в файле (то есть порядок, в котором они создаются) не имеет значения.

В программе `csv2html.py` первой вызываемой функцией является функция `main()`, которая в свою очередь вызывает функции `print_start()` и `print_line()`. Функция `print_line()` вызывает функции `extract_fields()` и `escape_html()`.

Когда интерпретатор Python читает файл, он начинает делать это с самого начала. Поэтому сначала будет выполнен импорт (если он есть), затем будет создана функция `main()`, а затем будут созданы остальные функции – в том порядке, в каком они следуют в файле. Когда интерпретатор, наконец, достигнет вызова `main()` в конце файла, все функции, которые вызываются функцией `main()` (и все функции, которые вызываются этими функциями), будут определены. Выполнение обработки, как и следовало ожидать, начинается в точке вызова функции `main()`.

Рассмотрим все функции по порядку, начиная с функции `main()`.

```
def main():
    maxwidth = 100
    print_start()
    count = 0
    while True:
        try:
            line = input()
            if count == 0:
                color = "lightgreen"
            elif count % 2:
                color = "white"
            else:
                color = "lightyellow"
            print_line(line, color, maxwidth)
            count += 1
        except EOFError:
            break
    print_end()
```

Переменная `maxwidth` используется для хранения числа символов в ячейке. Если поле больше, чем это число, часть строки отсекается и на место отброшенного текста добавляется многоточие. Программный код функций `print_start()`, `print_line()` и `print_end()` будет приведен чуть ниже. Цикл `while` выполняет обход всех входных строк – это могут быть строки, вводимые пользователем с клавиатуры, но мы предполагаем, что данные будут перенаправлены из файла. Далее выбирается цвет фона и вызывается функция `print_line()`, которая выводит строку в виде строки таблицы в формате HTML.

```
def print_start():
    print("<table border='1'>")

def print_end():
    print("</table>")
```

Мы могли бы не создавать эти две функции и просто вставить соответствующие вызовы `print()` в функцию `main()`. Но мы предпочитаем выделять логику, так как это делает реализацию более гибкой, хотя в этом маленьком примере гибкость не имеет большого значения.

```

def print_line(line, color, maxwidth):
    print("<tr bgcolor='{0}'>".format(color))
    fields = extract_fields(line)
    for field in fields:
        if not field:
            print("<td></td>")
        else:
            number = field.replace(",", "")
            try:
                x = float(number)
                print("<td align='right'>{0:d}</td>".format(round(x)))
            except ValueError:
                field = field.title()
                field = field.replace(" And ", " and ")
                field = escape_html(field)
                if len(field) <= maxwidth:
                    print("<td>{0}</td>".format(field))
                else:
                    print("<td>{0:.{1}} ...</td>".format(field, maxwidth))
    print("</tr>")

```

Мы не можем использовать метод `str.split(",")` для разбиения каждой строки на поля, потому что запятые могут находиться внутри строк в кавычках. Поэтому мы возложили эту обязанность на функцию `extract_fields()`. Получив список строк полей (в виде строк без окружающих их кавычек), мы выполняем обход списка и создаем для каждого поля ячейку таблицы.

Если поле пустое, мы выводим пустую ячейку. Если поле было заключено в кавычки, это может быть строка или число в кавычках, содержащее символы запятой, например "1,566". Учитывая такую возможность, мы создаем копию поля без запятых и пытаемся преобразовать ее в число типа `float`. Если преобразование удалось, мы определяем выравнивание в ячейке по правому краю, а значение поля округляется до ближайшего целого, которое и выводится. Если преобразование не удалось, следовательно, поле содержит строку. В этом случае мы с помощью метода `str.title()` изменяем регистр символов и заменяем слово «And» на слово «and», устраняя побочный эффект действия метода `str.title()`. Затем выполняется экранирование специальных символов HTML и выводится либо поле целиком, либо первые `maxwidth` символов с добавлением многоточия. Простейшей альтернативой использованию вложенного поля замены в строке формата является получение среза строки, например:

```

print("<td>{0} ...</td>".format(field[:maxwidth]))

```

Еще одно преимущество такого подхода состоит в том, что он требует меньшего объема ввода с клавиатуры.

```

def extract_fields(line):
    fields = []
    field = ""
    quote = None

```

```

for c in line:
    if c in '\\"':
        if quote is None: # начало строки в кавычках
            quote = c
        elif quote == c: # конец строки в кавычках
            quote = None
        else:
            field += c
            # другая кавычка внутри строки в кавычках
            continue
    if quote is None and c == ",": # end of a field
        fields.append(field)
        field = ""
    else:
        field += c
        # добавить символ в поле
if field:
    fields.append(field) # добавить последнее поле в список
return fields

```

Эта функция читает символы из строки один за другим и накапливает список полей, где каждое поле – это строка без окружающих ее кавычек. Функция способна обрабатывать поля, не заключенные в кавычки, и поля, заключенные в кавычки или в апострофы, корректно обрабатывая запятые и кавычки (апострофы в строках, заключенных в кавычки, и кавычки в строках, заключенных в апострофы).

```

def escape_html(text):
    text = text.replace("&", "&amp;")
    text = text.replace("<", "&lt;")
    text = text.replace(">", "&gt;")
    return text

```

Эта функция просто замещает каждый специальный символ HTML соответствующей ему сущностью языка HTML. В первую очередь, конечно, мы должны заменить символ амперсанда и угловые скобки, хотя порядок не имеет никакого значения.

8. Упражнения

8.1. Изменение вывода символов Юникода

Измените программу `print_unicode.py` так, чтобы пользователь мог вводить в командной строке несколько разных слов и получать только те строки из таблицы символов Юникода, в которых содержатся все слова, указанные пользователем. Это означает, что мы сможем вводить такие команды:

```
print_unicode_ans.py greek symbol
```

Подсказка. Один из способов достижения поставленной цели состоит в том, чтобы заменить переменную `word` (которая может хранить `0`, `None` или строку) списком `words`. Не забудьте изменить информацию о порядке использования. В результате изменений не более десяти строк программного кода добавится и не более десяти строк изменится. Имя файла: `print_unicode_ans.py`.

8.2. Изменение `quadratic.py`

Измените программу `quadratic.py` так, чтобы она не выводила коэффициенты со значением `0.0`, а отрицательные коэффициенты выводились бы как `-n`, а не `+ - n`. Имя файла: `quadratic_ans.py`.

8.3. Использование функции `escape()`

Удалите функцию `escape_html()` из программы `csv2html.py` и используйте вместо нее функцию `xml.sax.saxutils.escape()` из модуля `xml.sax.saxutils`.

Подсказка. Для этого потребуется добавить одну новую строку (с инструкцией `import`), удалить пять строк (с ненужной функцией) и изменить одну строку (задействовать функцию `xml.sax.saxutils.escape()` вместо `escape_html()`). Имя файла: `csv2html_ans1.py`.

8.4. Добавление обработки параметров командной строки в `csv2html.py`

Измените программу `csv2html.py` еще раз и добавьте в нее новую функцию с именем `process_options()`. Эта функция должна вызываться из функции `main()` и возвращать кортеж с двумя значениями: `maxwidth` (типа `int`) и `format` (типа `str`). При вызове функция `process_options()` должна устанавливать `maxwidth` в значение по умолчанию `100`, а строку `format` – в значение по умолчанию `".0f"`, которое будет использоваться как спецификатор формата при выводе чисел. Если пользователь вводит в командной строке `-h` или `-help`, должно выводиться сообщение о порядке использования и возвращаться кортеж `(None, None)`. (В этом случае функция `main()` ничего делать не должна.) В противном случае функция должна прочитать аргументы командной строки и выполнить соответствующие присваивания. Например, устанавливать значение переменной `maxwidth`, если задан аргумент `maxwidth=n`, и точно так же устанавливать значение переменной `format`, если задан аргумент `format=s`. Ниже приводится сеанс работы с программой, когда пользователь затребовал инструкцию о порядке работы:

Terminal

```
> csv2html2_ans.py -h
usage:
csv2html.py [maxwidth=int] [format=str] < infile.csv > outfile.html
maxwidth - необязательное целое число. Если задано, определяет
максимальное число символов для строковых полей. В противном случае
используется значение по умолчанию 100.

format - формат вывода чисел. Если не задан, по умолчанию используется
формат ".0f".
```

А ниже приводится пример командной строки, в которой установлены оба аргумента:

```
Terminal  
> csv2html2_ans.py maxwidth=20 format=0.2f < mydata.csv > mydata.html
```

Подсказка. Не забудьте изменить функцию `print_line()` так, чтобы она использовала переменную `format` при выводе чисел – для этого вам придется передавать функции дополнительный аргумент, добавить одну строку и изменить еще одну строку. И это немного затронет функцию `main()`. Функция `process_options()` должна содержать порядка двадцати пяти строк (включая девять строк с текстом сообщения о порядке использования).

Имя файла: `csv2html2_ans2.py`.