ЛАБОРАТОРНАЯ РАБОТА

Тема: ***Числовые типы данных***

Цель работы:

Отчет: Отчет представляет собой документ Word (шаблон прилагается), в который последовательно из окна интерпретатора копируются команды и результаты выполнения для каждого задания.

Содержание

Использование интерпретатора в режиме калькулятора

Вычислении выражений

Простейшее применение интерпретатора — вычисление выражений. Для этого выражение вводится в командную строку интерпретатора. Вычисление происходит после нажатия клавиши ввода `Enter`.

В командной строке интерпретатора *перед выражением нельзя вставлять пробелы*, т.е. делать отступ (англ. *identation*). В Питоне отступы играют особую роль — это инструмент для управления структурой программного кода.

Вывод нескольких результатов

Если результаты вычисления выражений не запоминаются в переменных, то говорят, что интерпретатор используется в режиме калькулятора. В этом случае он автоматически показывает на консоли результат любого вычисленного им выражения.

Обычно предполагается, что в одной строке программы содержится одна инструкция. Однако на самом деле в строку можно поместить две или более команды, отделяя каждую инструкцию от следующей точкой с запятой.

Перед второй и последующими инструкциями в строке могут быть добавлены пробелы. Например,

```
>>> 1+2; 4*2
3
8
```

В строке задано два выражения и, соответственно, получено два результата, каждый из которых выведен интерпретатором на новой строке.

Замечание.

Пользоваться таким способом записи нужно только в крайнем случае. Размещение каждой инструкции в отдельной строке соответствует хорошему стилю программирования. Это позволяет сделать программу и её структуру более понятной, и тем самым уменьшить вероятность ошибок программирования.

в) Сохранение последнего результата

Интерпретатор автоматически сохраняет результат последнего вычисленного выражения в специальной переменной, обозначаемой *символом подчеркивания*. Она доступна программисту "только для чтения", т.е. программист не может менять хранящееся в ней значение.

```
>>> 3
>>> _
3
```

Этой переменной пользуются, когда в очередном выражении нужно использовать результат вычислений предыдущего.

Кроме того, переменная "подчеркивание" позволяет пошагово смоделировать процесс вычисления выражения. Это, в частности, помогает находить место ошибки, когда при вычисления выражения получается неверный результат или выдается сообщение об синтаксической ошибке.

ЗАДАНИЕ 1 (*Использование интерпретатора как калькулятора*)

Задание выполняется в IDLE.

1. Ввести в командную строку интерпретатора выражение `1+2` и выполнить команду (нажать `Enter`). Интерпретатор вычислит результат и выведет его на консоль.

Вернуть выполненную инструкцию (`Alt+p`) в командную строку и отредактировать её: добавить перед единицей один или несколько пробелов. Попытаться выполнить инструкцию. Что выводит интерпретатор?

2. Используя переменную "подчеркивание", вычислить факториалы `2!`, `3!`, ... `6!`. Для этого сначала получить значение `1*2`, а затем последовательно в каждой новой команде умножать предыдущий результат на очередной множитель.

Числовые типы данных

В Питоне используется два вида числовой информации: *целые* и *нецелые* (вещественные или с точкой) числа.

Обычно в языках программирования максимальная величина целого числа ограничена разрядностью процессора (длиной "машинного слова"). Но в Питоне никаких ограничений нет — при вычислениях с очень большими числами он незаметно переходит на специальную "длинную арифметику".

а) Способы записи целых чисел.

Тип данных целых чисел обозначается `int`. Целые числа и только их, можно записывать в программах, используя разные системы счисления.

Числа в десятичной системе записываются обычным образом. Чтобы указать, что число представлено в другой системе счисления, используются специальные префиксы из *нуля* и буквы, указывающие основание системы:

- для *двоичной* (**b**inary) системы — `0b` или `0B`,
- для *восьмеричной* (**o**ctal) системы — `0o` или `0O`,
- для *шестнадцатеричной* (he**x**adecimal) системы — `0x` или `0X`.

б) Способы записи чисел с точкой.

Числа с точкой (вещественные) — это числа, имеющие дробную часть. Им соответствует тип данных `float`.

Точка является основным признаком вещественного числа, а целая *или* дробная часть в записи числа необязательны, должна присутствовать хотя бы одна из них.

Отсутствующая часть автоматически дополняется нулем. Т.е. числа `.2` и `3.` будут восприниматься, соответственно, как `0.2` и `3.0`.

Вообще, тип числа интерпретатор определяет по наличию или отсутствию десятичной точки.

В Питоне диапазон допустимых вещественных чисел широкий, но ограниченный. Границы диапазона приблизительно от 10^{-307} до 10^{+307}.

Если число больше максимально допустимого, то возникает ошибка "переполнение" (`overfow`). Если число меньше минимально допустимого, то оно считается нулем ("машинный нуль").

Научная нотация

В науке и технике приходится оперировать с числами, например, мировыми константами, имеющими очень большие или, наоборот, очень малые по модулю значения. При этом количество значащих цифр в них относительно небольшое.

Записывать такие числа обычным образом неудобно. Вместо этого применяется экспоненциальная форма (scientific notation, "научная нотация"), в которой запись числа строится из мантиссы и порядка. *Мантисса* (в общем случае вещественное число), умножается на степень числа `10`, которая называется *порядком*.

Обычно число преобразуется так, чтобы мантисса содержала одну цифру в целой части, т.е. принимала значения от `1.0` до `10.0`.

В программах основание степени `10` заменяется латинской буквой `e` или `E` (от англ. ***e****xponent*), за которой записывается величина порядка. Например, гравитационная постоянная Ньютона $6{,}6742867 \cdot 10^{-11}$ в программном коде представляется как `6.6742867e-11`.

Экспоненциальная форма применяется также, когда нужно коротко записать целое число с большим числом нулей. Например, масса Земли оценивается как 66000000000000000000 тонн. Из-за обилия нулей по этой записи трудно оценить порядок величины (6,6 секстиллионов). Научная нотация не только более компактна, но и более информативна: `6.6e21`.

Диапазон и *точность* представляемых в Питоне чисел с точкой ограничены: для представления мантиссы используется не более `15` значащих цифр, а порядок находится в пределах от 10^{-307} до 10^{+307}.

Если порядок числа *в научной нотации* больше максимально допустимого, то возникает переполнение. "Переполненное" значение трактуется Питоном как бесконечность и выводится в виде `inf` (от `infinity` — бесконечность).

Если величина числа в научной нотации меньше минимально допустимого, то оно обнуляется ("машинный нуль").

Определение типа данных

Чтобы определить тип, к которому принадлежит объект данных, в Питоне используется встроенная функция

```
type(выражение)
```

Её можно применять, в том числе, и к выражениям.

Тип значения, определяемый функцией, выводится в виде служебного слова `class` и имени типа (класса). Например,

```
>>> type(1+2)
<class 'int'>
```

ЗАДАНИЕ 2 (*Представление чисел*)

1. Заданное преподавателем натуральное число X последовательно представить в трех системах счисления:

– сначала перевести из десятичной системы в шестнадцатеричную,

– полученный результат записать в двоичной системе,

– а из двоичной системы в преобразовать в восьмеричную.

Используя префиксы соответствующих систем счисления, записать полученные значения по правилам Питона.

Проверить результаты перевода числа, последовательно вычисляя три выражения. В каждом выражения используется одно из полученных представлений, из которого вычитается значение `X` в десятичной системе.

2. Число "один миллион" можно представить в обычной форме, а можно с помощью научной нотации, например, сделав мантиссу равной `1`. Используя функцию `type()` определить, влияет ли на тип данных, к которому Питон относит число, форма его записи (применить `type()` к двум вариантам записи миллиона).

Операторы и операнды

Арифметические выражения составляются из чисел, соединенных знаками операций. Знак операции называется *оператором*, а данные, с которыми выполняется операция — *операндами*.

Если операция выполняется с одним операндом, то оператор называется *унарным* (от лат. "*единственный*").

Если операция выполняется с двумя операндами, то оператор называется *бинарным* (от лат. "*представленный двумя компонентами*").

Операторы сложения, умножения и деления (`+`, `*`, `/`) относятся к бинарным. А какую роль играет "`минус`", зависит от контекста.

Например, "`минус`" в выражении `2-3` — бин*арный* оператор вычитания. Но в выражении `-3` тот же "`минус`" выступает уже как ун*арный* оператор смены знака.

Арность оператора (количество операндов) играет роль при определении порядка действий. Например, для унарного и бинарного "минуса" определены разные уровни приоритета.

Операции с числами

В выражениях с числами могут использоваться различные арифметические операторы:

Операция	***Оператор***	***Пример***	***Результат***
Сложение	`+`	`34+1`	`35`
Вычитание	`-`	`34.0-0.1`	`33.9`
Умножение	`*`	`300*30`	`9000`
Деление	`/`	`3/8`	`0.375`
Деление с округлением вниз	`//`	`17//3`	`5`
Остаток от деления	`%`	`17%3`	`2`
`divmod` (сочетание `//` и `%`)		`divmod(17,5)`	`(5,2)`
Возведение в степень	`**`	`4**0.5`	`2.0`

Имеется три операции, связанные с делением.

Деление в обычном понимании выполняется с помощью оператора `/`. Даже если делимое делится на делитель без остатка, результат будет вещественным числом:

```
>>> 15/3
5.0
```

Операция `//` называется делением "с округлением вниз" (целочисленное деление). Её результатом является ближайшее к результату обычного деления *целое* число, которое *меньшее* этого результата.

Операция `%` используется для получения остатка от деления (деление по модулю).

Все три операции деления применимы как к целым числам, так и к числам с точкой.

Однако, хотя операцию `%` можно выполнять с любыми числами, рекомендуется применять её *только* к *целым положительным числам*. Для отрицательных или нецелых чисел трактовка результата усложняется, а неправильное толкование действий может стать причиной ошибки в программе.

Замечание

Для сложной работы с вещественными числами имеется специальный модуль math, упоминавшийся в лабораторной работе №1. Подключив его специальным образом, можно применять тригонометрические функции, логарифмы и т.п. Помимо этого, в модуле содержится определение двух констант: e и π.

При вычислениях с вещественными числами неизбежны потери точности. Интересный факт: при повторных операциях умножения и деления погрешность растет умеренно, а повторное сложение или вычитание погрешность увеличивают существенно.

ЗАДАНИЕ 3 (*Простые действия с числами*)

1. Дана последовательность команд для интерпретатора:

```
>>> 0.1
>>> _ + _ + 0.1
```

Что должно получиться в результате вычислений?

Последовательно выполнить команды и проверить, совпадает ли полученный результат с ожидаемым? Дать объяснение.

2. Вычислить выражения `15/2`, `15//2`, `-15//2`, `15.0//2` и объяснить результаты.

3. Студент может потратить на завтрак 85 рублей. Вычислить, сколько пирожков стоимостью 19 рублей он может купить и сколько денег у него останется?

4. Чтобы убедиться, что предсказать результат вычисления остатка от деления на отрицательное число не так просто, вычислить выражения `7//-3` и `7%-3` и попытаться дать объяснения результатам.

Старшинство операторов

Как и в большинстве других языков программирования, в языке Python можно создавать сложные выражения, объединяя в одной инструкции несколько операторов. При этом возникает проблема, известная из алгебры: в каком порядке нужно выполнять действия?

Когда интерпретатор Python встречает выражение, содержащее более одного оператора, он разбирает выражение на отдельные элементы данных и операторы. А затем, в соответствии с *правилами старшинства операторов,* определяет порядок вычислений.

В случае равных приоритетов операторов вычисления идут в порядке слева направо.

Из рассматриваемых в лабораторной работе операций самый высокий приоритет у операция "точка" (обращения к свойству), которая будет применяться для комплексных чисел, затем идет возведение в степень, а самый низкий приоритет у сложения и вычитания.

Для изменения порядка выполнения операторов используются *круглые скобки*. Python всегда в первую очередь вычисляет подвыражения в круглых скобках, а затем использует их результаты в объемлющем выражении.

Но скобки удобно применять и в тех сложных случаях, когда у программиста есть сомнения в порядке вычислений. Тогда, чтобы исключить возможность ошибки, лучше явно задать порядок вычислений с помощью скобок.

Приоритеты некоторых операторов

В таблице приведены приоритеты операций, используемых в лабораторной работе, *в порядке возрастания* их старшинства.

В одной строке расположены операторы с равным приоритетами.

Операторы	Смысл операции
`+, -`	Сложение и вычитание
`*, /, //, %`	Умножение, деление, остаток
`+x, -x`	Определение и смена знака (унарные)
`**`	Возведение в степень
`x.свойство`	Обращение к свойству

ЗАДАНИЕ 4 (*Старшинство операторов*)

1. Объяснить, как в Питоне будет интерпретироваться выражение `5+-2` и почему. Расставить скобки.

2. В выражении `11/+2%4+2%3` расставить скобки в соответствии со старшинством операторов. Объяснить, как получается результат вычислений.

Комплексные числа

Питон имеет встроенную поддержку комплексных чисел. Но мнимая единица обозначается не так это принято в математике буквой *i*, а строчной `j` или заглавной `J`.

Комплексное число состоит из действительной и мнимой частей, коэффициенты которых могут быть целыми числами или числами с точкой.

Комплексные числа можно задавать двумя способами: непосредственно (например, `2+1j` или `2+1J`) или же функцией `complex`, которой передаются коэффициенты числа: `complex(2,1)`. Эти два способа полностью эквивалентны.

Буква `j` воспринимается как мнимая единица, если она записана *сразу после числа* без дополнительных знаков и пробелов. Из этого правила следует, что число "мнимая единица" должно записываться как `1j`, а не просто `j`.

В памяти комплексное число представляется упорядоченной парой *вещественных* чисел, даже если реальная и мнимая части заданы целыми числа. Поэтому `1+2j` и `1.0+2.0j` будут храниться одинаково.

В Питоне при вычислениях с комплексными числами есть возможность получать значения их действительной и мнимой частей, находить модуль числа или сопряженное значение.

Выделить действительную и мнимую часть числа можно следующим образом:

```
>>> (1+2j).real
1.0
>>> (1+2j).imag
2.0
```

Действительная и мнимая часть рассматриваются как свойства комплексного числа. Операция "точка" используется для обращение к свойствам объектов, в данном случае — к свойствам комплексного числа.

А с помощью метода `conjugate()`, которым также обладают комплексные числа, можно получить сопряженное число:

```
>>> (1+2j).conjugate()
(1-2j)
```

Модуль комплексного числа вычисляется с помощью встроенной функции `abs()`:

```
>>>abs(1+1j)
1.4142135623730951
```

ЗАДАНИЕ 5 (*Операции с комплексными числами*)

1. Какая ошибка допущена в следующем выражении:

```
3 + j + 2 - 5j
```

2. Взять два произвольных комплексных числа.

Проверить, какие из разрешенных для действительных чисел операций возможны также и для комплексных чисел:

`*` — умножение на целое число и на комплексное число,

`/` — деление на целое, вещественное и комплексное число,

`//`, % — с делителем в виде целого числа,

** — возведение в целую, вещественную и комплексную степень.

3. Взять комплексное число, например, `3-4j`, и вычислить его модуль. Для этого сначала пошагово выполнить следующие вычисления (использовать переменную "подчеркивание"):

– комплексное число умножается на сопряженное (использовать `conjugate()`),

– из результата умножения извлекается квадратный корень (с помощью операции возведения в степень) и

– из результата извлечения корня выделяется действительная часть (`real`).

Затем, учитывая приоритеты операций, построить для вычисления модуля *одно выражение*, в котором собраны все эти действия.

Чтобы проверить правильность вычислений, найти модуль того же числа с помощью встроенной функции `abs(компл_число)`.

4. Выяснить, что происходит, если вызвать функцию `complex` с одним аргументом.

Выражения со смешанными типами операндов

Бинарные операции над числами допустимы только тогда, когда оба операнда *принадлежат одному и тому же типу*. При этом обычно не возникает вопрос о типе результата, так как он однозначно определен для данного типа. Например, сложение двух значений типа `int` даст в результате `int`, а их деление / — значение `float`.

Но даже в самых простых выражениях могут встречаться операнды разных типов: комплексный и целый, комплексный и вещественный, вещественный и целый.

Тогда перед выполнением операции необходимо согласовать типы операндов, сделать их одинаковыми. Для этого выполняется операция *приведения типа*.

Преобразование (приведение) *типа* — это унарная операция. В результате её выполнения будет получено значение операнда, представленное в виде, соответствующем другому типу данных.

Примеры приведение для чисел (стрелка в примерах к синтаксису Питона *не относится*!):

`1`→`1.0` — приведение целого к вещественному,

`1`→ `(1.0+0j)` или `1.0`→`(1.0+0j)` — преобразование целого или вещественного в комплексное,

`2.0` → `2` — приведение вещественного к целому.

Приведение типа бывает *явным* и *неявным*. Явные преобразования задаются программистом с помощью специальных инструкций (будут рассмотрены позже).

Неявные преобразования называются так, потому что они выполняются интерпретатором автоматически, скрыто от программиста.

С их помощью производится согласование типов операндов при вычислении выражений или выполнении операций сравнения. В языке Питон предусмотрены случаи, когда они будут выполняться.

Интерпретатор Python упорядочивает числовые типы по сложности следующим образом: *целые* числа проще, чем *числа с точкой*, которые, в свою очередь, проще *комплексных* чисел.

Общее правило при вычислении числовых выражений с операндами разных типов: значение операнда, который имеет более простой тип, преобразуется к типу более сложного. Соответственно, результат выполнения операции будет иметь тип более сложного операнда.

Например, когда в выражении участвуют *целое* и *вещественное* числа, то целое число преобразуется в число с точкой, операция выполняется по правилам для вещественных чисел, что дает в результате *число с точкой*.

Если одним из операндов выражения является *комплексное* число, а второй — *целое или вещественное*, то выполняется приведение второго операнда к комплексному типу, и результат будет иметь *комплексный* тип.

ЗАДАНИЕ 6 (*Неявные преобразования типа*)

1. Сложить произвольное целое число с вещественным, и выяснить, какой тип имеет результат вычислений.

Повторить эти действия для случая, когда операнды имеют вещественный и комплексный тип.

2. Вычислить значение 3^{650}. Результатом будет длинное значение, имеющее порядок приблизительно 10^{310}.

Попробовать вычислить значение выражения `3**650-0.1`, которое явно немного меньше, чем в предыдущем случае. Проанализировать сообщение на консоли. Как объяснить возникшую проблему?

Вопросы для самоконтроля

1. Можно ли начать запись выражения в командной строке интерпретатора с пробела?

2. Каким образом в командной строке интерпретатора в новом выражения можно использовать результат вычисления предыдущего выражения?

3. Из каких символов при записи целых чисел состоят префиксы чисел, указывающие на основание системы счисления?

4. Что означает отсутствие в записи вещественного числа целой или дробной части?

5. Есть ли ограничения на величину целых и вещественных чисел, представимых в Питоне?

6. Как интерпретатор по записи числа определяет тип числа (целый или вещественный)?

7. В чем суть научной нотации и когда её применяют?

8. Что в записи выражения называется оператором и что операндами?

9. В чем отличие унарных и бинарных операторов?

10. Какими способами записываются комплексные числа?

11. Как получить действительную и мнимую части комплексного числа, его модуль и число, сопряженное с ним?

12. Что означает операция приведения типа?

13. Какие преобразования типа называют неявными? Для чего они применяются при вычислении выражений?

14. По каким правилам выполняются преобразования типа для бинарных операций?

15. На чем основан порядок вычислений, если в выражении присутствует несколько операторов? Как можно управлять этим порядком?