ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ PYTHON

(Python Beginning)

«Теперь он и тебя сосчитал» – сказал телёнок своей маме.
(Из сказки Альфа Прёйсена «Козлёнок, который умел считать до десяти»)

Урок 3. (Lesson 2) Основы программирования

Встроенные типы данных

Множество встроенных типов данных определяет множество задач обработки данных, которые могут быть успешно и легко решены не прибегая к сознанию вспомогательных программных инструментов. *Python* относится к категории универсальных языков программирования. Поэтому его разработчики постарались учесть тот ***«универсальный круг задач»***, с которыми пришлось столкнуться им и другим программистам в конце 80-х годов XX века. Модульная организация *Python* позволила организовать так называемую ***«базовую поставку»*** языка, которая позволяет решать ***условно универсальный круг задач***. Потребность в решении специальных задач удовлетворяется за счет импорта необходимых модулей, например модуля *numpy*, предназначенного для решения задач линейной алгебры.

Python предоставляет программисту мощную коллекцию встроенных типов данных. Большая часть встроенных типов данных языка *Python* сопоставима с множеством специальных типов данных, реализованных в стандартной библиотеке (*STL*) языка *C++*. Причем, если изучение *STL C++* обычно начинается не ранее чем через полгода после преступления к изучению языка, то в *Python* практически сразу начинают работать с типами данных, относящихся к категории контейнеров. Перечислим возможности, которые предоставляют программисту встроенные типы данных:

- ***Встроенные типы данных упрощают процесс создания программ.*** Для решения сложных задач можно использовать как собственные объекты, так и встроенные классы языка *Python* или интерфейсы языка *C*. Но использование уже имеющихся программных инструментов значительно снижает затраты времени на разработку. Пусть это и потребует от вас дополнительного времени на профессиональную подготовку, так как это ваша святая обязанность – профессиональный рост. Благодаря встроенным типам данных вы сможете выполнить значительно больший объем работы, нежели будете все начинать «с чистого листа».
- ***Встроенные типы данных – это компоненты расширения языка***. Помимо простых вычислительных задач, *Python* позволяет решать задач категории

когнитивной обработки данных: задачи, связанные с построением и моделированием нейронных сетей и многие другие.

- ***Встроенные типы данных*** являются плодом труда коллектива программистов, поэтому часто они ***более эффективны, чем созданные вручную структуры данных***. Для создания высокоэффективных программных инструментов требуется достаточно высокий уровень профессиональной квалификации и большой опыт работы. Это то, чего пока ещё нет у начинающего программиста. Стандартные типы данных проектировались и создавались с учетов базовой архитектуры ЭВМ и теорией вычислительных процессов. Этими знаниями, как правило, начинающие программисты тоже не владеют…
- ***Встроенные типы данных – это стандартная часть языка***. Стандарт языка определяет неизменный набор модулей и типов данных, входящих с состав «***базовой поставки***» языка. Вы можете сами создавать свои проблемно-ориентированные типы данных, но для решения большинства задач, скорее всего, хватит стандартных типов. И это замечательно, что в эту «базовую поставку» `Python` входят средства (классы) для разработки графического интерфейса пользователя!

Один студент – старшекурсник похвастался, что выучил Python за две недели. Возможно, он успел за две недели ознакомиться с синтаксисом языка, но для нормального изучения Python нужно не менее полугода…

Кратко перечислим базовые типы данных языка Python:

- числовые типы;
- строки;
- списки;
- словари;
- кортежи;
- файлы;
- множества;
- специальные типы данных (t`ype`, `NoneType`, `bool`);
- функции, модули, классы;
- типы, имеющее отношение к реализации.

Числовые типы данных

Язык программирования Python версии 3.x включает в себя следующие числовые типы данных:

`int` – целочисленный тип;

`float` – вещественный тип;

`complex` – комплексно сопряженные числа.

Отдельно следует отметить тот факт, что при разработке *Python* старались реализовать целочисленный тип таким образом, чтобы число в памяти хранилось в виде строки. Данный подход позволил обойти ограничение на диапазон значений, присущий представлению целых чисел в других языках программирования, таких как C и C++. Это позволяет реализовывать на Python «длинную арифметику» за счет встроенного базового типа *int*. Повторим пример из первого урока и вычислим значение 100^{100} :

```
>>> 100 ** 100
10000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000
>>>
>>> type(100 ** 100)
<class 'int'>   # тип результата выражения - int
>>>
```

На операцию возведения в степень для вещественных чисел распространяются ограничения по размеру переменных, а на возведение целого в целую степень такого ограничения нет.

```
>>> 1.2 ** 3.4  # вещественное в вещественной степени
1.858729691979481
>>> 100.01 ** 100.01
1.0576528937261317e+200  # максимальный размер - +3.4e+308
>>> 1000.001 ** 1000.001
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    1000.001 ** 1000.001
OverflowError: (34, 'Result too large')
>>> len(str(1000 ** 1000))
3001
>>>
>>> 1000 ** 1000
10000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
   .    .    .    (для экономии места пропущен блок нулей)    .    .    .
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000
>>>
```

По материалам второго урока вы уже знаете, что язык программирования *Python* позволяет программисту представлять целые числа в 2-ичном, 8-ричном, 10-чном и 16-чном форматах. Механизм отображения не влияет не способ хранения чисел в памяти.

Создать объект числового типа в программе можно «***обычным способом***»: объявить переменную-ссылку и связать её с объектом в памяти, то есть инициализировать.

```
>>> A = 12; B = 12.34567
>>> type(A); type(B)
<class 'int'>
<class 'float'>
>>>
```

Но даже здесь разработчики *Python* добавили «***голландский синтаксический сахар***», связанный с оптимизацией памяти.

```
>>> var01 = 12
>>> var02 = 12
>>> var01 is var02  # проверка: var01 это var02?
True                # Да, ссылки связаны с одним объектом в памяти
>>>
```

Если ссылки связываются с одинаковыми значениями, то объект в памяти создается один. Но если по одной из ссылок выполняется изменение объекта, но сразу же создается новый «дополнительный» объект в памяти.

```
>>> var01 +=1
>>> var01
13
>>> var02
12
>>> var01 is var02  # проверка: var01 это var02?
False               # Ложь, они связаны с разными объектами
>>>
```

В то же время Python не оптимизирует распределение памяти при выполнении операций над вещественными числами.

```
>>> var01 = 1.001
>>> var02 = 1.001
>>> var01 is var02  # проверка: var01 это var02?
False               # Ложь, они связаны с разными объектами
>>>
```

В Python реализованы «***почти такие же***» как и в C/C++ правило вычисления типа результата выражения. Но, по «***математической необходимости***», *Python* автоматически преобразует целочисленное значение к вещественному типу.

```
>>> type(12/2)
<class 'float'>
>>> type(12)
<class 'int'>
```

```
>>> type(2)
<class 'int'>
>>> type(12 + 2)
<class 'int'>
>>> type(12 * 2)
<class 'int'>
>>> type(12 - 2)
<class 'int'>
>>>
```

Фактически, *Python* – разработчики «***исправили косяк целочисленной математики***», реализованной в C/C++: при делении целого на целое результат в C/C++ целого типа, к чему так долго не могли привыкнуть первокурсники, начинающие изучать «***великий и могучий***» *ANSI C*. И ведь это был не косяк, а просто реализация набора операций, использующих блок центрального процессора, связанного с целочисленной арифметикой. В середине 70-х годов XX века *ANSI C* позиционировался, как макроассемблер…

```
>>> print(12/2, type(12/2))
6.0 <class 'float'>
>>> print(12//2, type(12//2))
6 <class 'int'>
>>>
```

Для «спасения» целочисленной арифметики была введена ***операция целочисленного деления*** – `'//'`.

В *Python* при выполнении операций над вещественными числами необходимо учитывать ограничения точности вычислений, связанную с понятием «машинного нуля», присущую ***числам с плавающей точкой***.

```
>>> 0.3 - 0.1 - 0.1 -0.1
-2.7755575615628914e-17
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
0.9999999999999999
>>> sum = 0.0
>>> for i in range(10): sum += 0.1

>>> print(sum)
0.9999999999999999
>>>
```

Для исправления данной проблемы реализован ***класс чисел с фиксированной точной*** – *Decimal*.

```
>>> from decimal import Decimal
>>> Decimal('0.3') - Decimal('0.1') - Decimal('0.1') - Decimal('0.1')
Decimal('0.0')
>>>
```

Как вы видите, для получения доступа к возможностям класса *Decimal*, необходимо импортировать содержащий его модуль *decimal*. Первую строчку использованного примера можно дословно перевести, как:

"Из_модуля decimal импортировать класс_Decimal".

Это позволяет минимизировать объем создаваемого исполняемого модуля за счет включения в него только необходимых программных компонент. Подобного эффекта можно было достичь и в *ANSI C* за счет использования специальных опций линковщика, здесь же, в *Python*, все стало значительно проще.

«***Переведём***» и пример формирования суммы десятых с использованием данного класса.

```
>>> sum = Decimal('0.0')
>>> for i in range(10): sum += Decimal('0.1')

>>> print(sum)
1.0
>>>
```

Кстати, даже при работе с этими специальными типами можно «***наступить на детские грабли***» =)

```
>>> Decimal('0.1') + Decimal('0.1') + Decimal('0.1') - Decimal('0.3')
Decimal('0.0')
>>> Decimal(0.1) + Decimal(0.1) + Decimal(0.1) - Decimal(0.3)
Decimal('2.77555756156515654042363166 8E-17')
>>>
```

Нетрудно заметить, что в случае передачи конструктору *Decimal()* экземпляра класса *float*, мы сами невольно искажаем истинное значение и привносим погрешность, связанную с плавающей точкой... Один из вариантов решения, представленный первым выражением, может быть использование преобразования *float* в *str* – лишняя вспомогательная операция, которая тоже «кушает процессорные такты», но обеспечивает точность.

```
>>> Decimal(str(0.1)) + Decimal(str(0.1))\
           + Decimal(str(0.1)) - Decimal(str(0.3))
Decimal('0.0')
>>>
```

Выбор варианта составления выражения для формирования значения аргумента конструктора *Decimal()* зависит от конкретной задачи: *Decimal(str(floar_argement))* или *Decimal(str_argument)* – например, при чтении данных из файла вы все равно считываете строковое представление вещественных чисел, то же самое происходит при чтении данных их файла базы данных. Подумайте, нужно ли сначала преобразовывать данные, прочитанные из

файла, в число, чтобы потом снова преобразовать их в строку, дабы получить данные специального типа?

Несколько примеров на использование класса *decimal*:

```
>>> from decimal import *
>>> getcontext().prec = 6  # управление фиксированной точкой
>>> Decimal(1) / Decimal(7)
Decimal('0.142857')
>>> getcontext().prec = 21
>>> Decimal(1) / Decimal(7)
Decimal('0.142857142857142857143')
>>>
```

Метод *getcontext*() позволяет управлять точность представления – числом знаков после «***фиксированной точки***».

```
>>> getcontext()
Context(prec=21, rounding=ROUND_HALF_EVEN, Emin=-999999, Emax=999999,
capitals=1,   clamp=0,   flags=[Inexact,   FloatOperation,   Rounded],
traps=[InvalidOperation, DivisionByZero, Overflow])
>>>
>>> x = Decimal('0.3')
>>> y = Decimal('0.666')
>>> z = x / y
>>> z
Decimal('0.450450450450450450450')
>>> getcontext().prec = 10
>>> z  # изменение точности не влияет на уже созданный объект
Decimal('0.450450450450450450450')
>>> z = x / y
>>> z
Decimal('0.4504504505')
>>>
>>> z = x + y; z
Decimal('0.966')
>>> z = x * y; z
Decimal('0.1998')
>>> z = x * y; print(z)
0.1998  # числа выводятся в обычном формате вещественных чисел
>>> z = x + y; print(z)
0.966
>>>
>>> decimal.Decimal('Infinity')  # Бесконечность
Decimal('Infinity')
>>> print(decimal.Decimal('Infinity'))
Infinity
>>>
>>> decimal.Decimal('Nan')  # не число
Decimal('NaN')
```

```
>>>
```

А еще в классе *decimal* есть перегруженный конструктор, позволяющий формировать экземпляр класса на основании набора цифр, представляющих число и значения экспоненты – целочисленной десятичной степени.

```
>>> a = decimal.Decimal((0,(2,3,4,5,6,7),-1))
>>> b = decimal.Decimal((1,(2,3,4,5,6,7),-1))
>>> c = decimal.Decimal((0,(2,3,4,5,6,7),-2))
>>> d = decimal.Decimal((1,(2,3,4,5,6,7),-2))
>>> e = decimal.Decimal((0,(2,3,4,5,6,7),5))
>>> a, b, c, d, e
(Decimal('23456.7'),     Decimal('-23456.7'),     Decimal('2345.67'),
Decimal('-2345.67'), Decimal('2.34567E+10'))
>>>
```

В этом случае конструктор принимал в качестве параметра кортеж – неизменяемый набор параметров, первый элемент которого представляет сигнал знака числа, второй элемент – кортеж, содержащий цифры, а третий – целочисленную степень экспоненты.

Кроме того, для работы с числовыми данными в *Python* введен класс рациональных чисел – *fractions*. Для нас важно, что для данного класса перегружены все обычные математические операции. Конструктор класса *fractions* определен для трех вариантов параметров:

```
frasctions.Fraction(числитель=0[, знаменатель=1])
frasctions.Fraction(дробь)
frasctions.Fraction(строка)
```

При необходимости, экземпляр класса *fractions*, может быть преобразован к классу *float* (использование конструктора преобразования).

```
>>> from fractions import Fraction
>>> Fraction(3/4)
Fraction(3, 4)
>>> float(Fraction(3,4))
0.75
>>> Fraction("1/3")
Fraction(1, 3)
>>> Fraction(12,13)
Fraction(12, 13)
>>> float(Fraction(12/13))
0.9230769230769231
>>>
```

Учтите, если вам необходимо разрабатывать приложение, реализующее финансовые и/или научные расчеты, и вы сильно зависите от точности

представления вещественных чисел, то для представления числовых данных необходимо использовать классы `decimal` и/или `fractions`!

Говоря языком терминов языка *Java*, можно назвать классы *decimal* и *fractions* классами-обертками.

Специальные методы для работы с числовыми данными

Для работы с простыми числовыми типами данных в Python имеются специальные методы, которые по функционалу можно отнести к двум группам: группа методов – конструкторов преобразования и группа методов, выполняющих роль математических операций.

- **Конструкторы преобразования**

Конструктор преобразования предназначен для преобразования объекта одного типа в его ближайший аналог – эквивалент другого типа. Например, преобразование из строкового типа в целочисленный в *ANSI C* выполнялось с помощью функции *atoi*(), в языке *C++* для таких задач перегружаются методы *static_cast<тип>()* и *dinamic_cast<тип>()*. В Python пошли по пути упрощения синтаксиса. Простые примеры по использованию таких конструкторов преобразования мы уже использовали, например:

```
>>> int('12')  # преобразование из str в int
12
>>>
```

Следует отметить, что перегружены далеко не все конструкторы преобразования и незнание правил их использования приводит к выбросу объекта исключения. Например:

```
>>> int("12.3")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    int("12.3")
ValueError: invalid literal for int() with base 10: '12.3'
>>>
```

Подробно рассмотрим синтаксис данных методов.

int([<*Объект*>[,<*Система_счисления*>]]) – преобразует объект в целое число. Вторым необязательным параметром является целочисленное значение системы счисления. По умолчанию используется 10-чная система счисления. Параметр <*Объект*> может быть числом или строковой записью числа.

```
>>> int(12.3)
12
```

```
>>> int("14"), int("14", 8), int("14", 16)
(14, 12, 20)
>>> int(14, 16)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    int(14, 16)
TypeError: int() can't convert non-string with explicit base
>>>
```

Обратите внимание, что в случае, если указывается параметр – значение системы счисления, параметр – объект должен быть строкой! Допускается использовать любые значения для значения системы счисления при условии корректности записи преобразуемого значения к его 10-чному эквиваленту.

```
>>> int("12", 3), int("12", 5), int("12", 7)
(5, 7, 9)
>>>
>>> int("9", 8)
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    int("9", 8)
ValueError: invalid literal for int() with base 8: '9'
>>>
>>> int("0b10101011", 2)
171
>>> int("0b10101")
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    int("0b10101")
ValueError: invalid literal for int() with base 10: '0b10101'
>>>
```

И не забывайте о значении системы счисления по умолчанию. ***Да минует вас Traceback!*** ☺☺☺

```
>>> int("111", 3), int("0111", 3), int("000111", 3)
(13, 13, 13)
>>> int("0b111", 3)  # в записи используется префикс 2-ичного формата
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    int("0b111", 3)
ValueError: invalid literal for int() with base 3: '0b111'
```

Методы преобразования можно назвать отчасти умными – они умеют игнорировать лишние ведущие нули, но они строго учитывают наличие префикса системы счисления для формирования внутренних правил интерпретации записи числа.

float([<Объект>]) – преобразует объект, который может быть числом или строкой, в вещественное число.

```
>>> float(2), float('2'), float(2.0), float("2.")
(2.0, 2.0, 2.0, 2.0)
>>> float(0b010101), float(0xFF)
(21.0, 255.0)
>>> float(0b0101)
5.0
>>> float('0b0101')  # строка должна соответствовать 10-чному формату
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    float('0b0101')
ValueError: could not convert string to float: '0b0101'
>>>
>>> float(23/17)  # преобразуем результат выражения
1.3529411764705883
>>>
>>> float(100**100)  # а если аргумент очень большой...
1e+200
>>>
>>> float(1**-100), float(10**-100)
(1.0, 1e-100)
>>>
```

bin(<Объект>) – преобразует объект целого типа в строку его двоичного представления. Данный метод определен только для целочисленных аргументов.

```
>>> type(bin(12)), bin(12), bin(0xFF), bin(0o77)
(<class 'str'>, '0b1100', '0b11111111', '0b111111')
>>>
>>> bin('12')  # использован параметр - строка, ошибка!
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    bin('12')
TypeError: 'str' object cannot be interpreted as an integer
>>>
```

oct(<Объект>) – преобразует объект целого типа в строку его 8-ричного представления.

```
>>> type(oct(77)), oct(123), oct(0xaa), oct(0o1234)
(<class 'str'>, '0o173', '0o252', '0o1234')
>>>
```

hex(<Объект>) – преобразует объект целого типа в строку его 16-ричного представления.

```
>>> type(hex(255)), hex(255), hex(0xff), hex(0o7777)
(<class 'str'>, '0xff', '0xff', '0xfff')
```

```
>>> hex(1e+2)  # ошибка: параметр – вещественное значение
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    hex(1e+2)
TypeError: 'float' object cannot be interpreted as an integer
>>> hex(int(1e+2))  # а так можно
'0x64'
>>> hex(12+4*3-1)  # параметром является результат выражения
'0x17'
>>> hex(16/2)  # не забываем об особенности операции деления...
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    hex(16/2)
TypeError: 'float' object cannot be interpreted as an integer
>>>
```

*В языке Python выполняется более строгая проверка типов параметров методов, нежели в C/C++. В C/C++ вы формируете выражение и, зачастую, считаете, что компилятор сам как-нибудь решит задачу преобразования типов, но не всегда такой подход приводит к ожидаемым результатам. Такой подход ведет к переполнению разрядной сетки, выходу за пределы допустимого диапазона и прочим «**прелестям жизни**». Python сразу **пытается приучить** программиста к аккуратности… ☺ ☺ ☺*

- **Методы – математические операции**

round(<Объект>[,<Точность>]) – в качестве параметра *<Объект>* метод принимает вещественное значение, возвращает вещественное значение, равное округленному по правилам математики: до ближайшего меньшего целого для чисел с дробной частью меньше 0.5, или до ближайшего большего целого для чисел с дробной частью больше 0.5. Если дробная часть равна 0.5, то округление производится до ближайшего четного числа.

```
>>> round(0.1), round(0.49), round(0.5), round(0.500), round(0.51)
(0, 0, 0, 0, 1)
>>> type(round(0.9999))
<class 'int'>
>>>
>>> from fractions import Fraction
>>> round(Fraction('1/3'))
0
>>> round(Fraction('1/3')+Fraction('1/3'))
1
>>>
>>> round(float("12.345"))
12
```

```
>>> round("12.345")  # передан не числовой параметр
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    round("12.345")
TypeError: type str doesn't define __round__ method
>>>
>>> round(12.34567890123456789, 5)  # округление с заданной
12.34568
>>> round(12.34567890123456789, 5)
12.34568
>>> round(Fraction('1/3'), 4)  # не все так просто, как кажется...
Fraction(3333, 10000)  # получили объект класса Fraction, а не float
>>> round(float(Fraction('1/3')),4)
0.3333
>>>
>>> from decimal import Decimal
>>> round(Decimal('3.4')/Decimal('4.5'))
1
>>> round(Decimal('3.4')/Decimal('4.5'), 7)
Decimal('0.7555556')  # эквивалент использования getcontext().perc
>>> round(float(Decimal('3.4')/Decimal('4.5')), 7)
0.7555556
>>>
```

abs(<Объект>) – метод возвращает абсолютное значение для параметра *<Объекта>*, который должен быть числом.

pow(<Число>, <Степень> [,<Остаток_от_деления>]) – возвращает значение, равное $<Число>^{<Степень>}$. Если указан третий параметр, то возвращается остаток от деления.

```
>>> pow(2, 8), 2 ** 8, pow(10, 3), 10 ** 3
(256, 256, 1000, 1000)
>>> pow(2, 8, 10), (2 ** 8) % 10
(6, 6)
>>>
```

При выполнении данной операции настоятельно рекомендуется не забывать математические правила вычисления степени числа, в противном случае можно вызвать исключение.

```
>>> pow(0.000000, 2)
0.0
>>> pow(0.0000000, -2)  # деление на ноль
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    pow(0.0000000, -2)
ZeroDivisionError: 0.0 cannot be raised to a negative power
>>>
```

min(<Список_параметров>) – возвращает минимальное значение из списка. Мы ещё не знакомились с контейнерами последовательного доступа – списками и кортежами, но начнем их простейшее использование. Под списком подразумевается последовательность объектов (ссылок на объекты), разделенных символом запятой – это как список параметров в *C/C++*. Но нам уже важно видеть различия между C/C++ и Python.

```
>>> min(19, 23, 98, 0xA, 0o7777, 1.2e+3, 11)
10
>>> min('Hello', 'Mr.Braun')
'Hello'
>>>
>>> a = (1,2,3,4,5)
>>> min(a)
1
>>> min(*a)  # параметр - контейнер
1
>>> b = {1: 'a', 2: 'b'}
>>> min(b)
1
>>> min(*b)  # параметр - контейнер
1
>>> b[min(b)]
'a'
>>> b[min(*b)]
'a'
>>>
```

max(<Список_параметров>) – возвращает максимальное значение из списка.

```
>>> max(19, 23, 98, 0xA, 0o7777, 1.2e+3, 11)
4095
>>> max('Hello', 'Mr.Braun', 'Black Star', 'Zoom')
'Zoom'
>>> b = {1: 'a', 2: 'b', 3: 'zerro', 4: 'Mazzy'}
>>> max(b)  # сравниваются значения ключей!
4
>>> b[max(b)]  # обратились к элементу с ‘наибольшим’ ключом
'Mazzy'
>>> max(*b)
4
>>>
```

sum(<Последовательность> [,<Начальное_значение>]) – возвращает сумму значений элементов последовательности. Аргумент должен быть именно списком или кортежем. Правила языков *C/C++* приводят к генерации исключения.

```
>>> a = (1,2,3,4,5)  # Параметр a - это кортеж
```

```
>>> sum(a)
15
>>> sum((1,2,3,4,5,6,7,8,9))
45
>>> sum(1,2,3,4,5,6,7,8,9)  # список параметров по правилам C/C++
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    sum(1,2,3,4,5,6,7,8,9)
TypeError: sum expected at most 2 arguments, got 9
>>>
>>> list1 = [1,2,3,4,5,6,7,8]  # list1 - список
>>> s0 = 9  # начальное значение для суммы
>>> sum(list1, s0)
45
>>>
```

divmod(x, y) – возвращает кортеж из двух значений (x//y, x%y).

```
>>> divmod(3, 5)  # 5 = 0 * 5 + 3
(0, 3)
>>> 3 // 5, 3 % 5
(0, 3)
>>> divmod(45, 7)
(6, 3)
>>> 45 // 7, 45 % 7
(6, 3)
>>> divmod(123.4567, 98.76)
(1.0, 24.696699999999993)
>>> 123.4567 // 98.76, 123.4567 % 98.76
(1.0, 24.696699999999993)
>>>
>>>
```

Средства модуля math

Модуль *math* предоставляет весь необходимый инструментарий для организации вычислений в Python. Подробную информацию можно получить на официальном сайте Python-сообщество [1]. Для получения доступа к его компонентам модуль нужно импортировать.

```
>>> import math
```

Краткую справку о средствах модуля можно получить из встроенной справочной системы:

```
>>> help(math)
Help on built-in module math:
  .    .    .
```

Кратко вспомним о математических константах, определенных в данном модуле:

```
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>> math.tau  # Введено в версии 3.6, ранее не поддерживается!
Traceback (most recent call last):
  File "<pyshell#53>", line 1, in <module>
    math.tau
AttributeError: module 'math' has no attribute 'tau'
>>> math.inf
inf
>>> math.nan
nan
>>>
```

Комплексно сопряженный тип тоже является встроенным типом данных. Средства для работы с этим типом определены в модуле cmath.

```
>>> var1 = 1 + 23j
>>> type(var1)
<class 'complex'>
>>> var2 = var1 + 12
>>> var2
(13+23j)
>>> var1 * var2
(-516+322j)
>>>
>>> import cmath
>>> cmath.sqrt(-1)
1j
>>> cmath.exp(var1)
(-1.4483903167752235-2.300265547540635j)
>>> cmath.log(var1)
(3.1364385032730837+1.5273454314033659j)
>>> cmath.log(var1, 12)
(1.262195706040172+0.6146490177140744j)
>>> cmath.log10(var1)
(1.3621379348003944+0.6633176928186234j)
>>>
```

Задачи

1. Найти расстояние между двумя точками с заданными координатами x_1 и x_2 на числовой оси: $|x_2 - x_1|$.
2. Даны три точки A, B, C на числовой оси. Найти длины отрезков AC и BC и их сумму.

3. Даны три точки A, B, C на числовой оси. Точка C расположена между точками A и B. Найти произведение длин отрезков AC и BC.
4. Даны координаты двух противоположных вершин прямоугольника: (x_1, y_1), (x_2, y_2). Стороны прямоугольника параллельны осям координат. Найти периметр и площадь данного прямоугольника.
5. Найти расстояние между двумя точками с заданными координатами (x_1, y_1) и (x_2, y_2) на плоскости. Расстояние вычисляется по формуле $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
6. Даны координаты трех вершин треугольника: (x_1, y_1), (x_2, y_2), (x_3, y_3). Найти его периметр и площадь, используя формулу для расстояния между двумя точками на плоскости (смотри предыдущую задачу). Для нахождения площади треугольника со сторонами a, b, c использовать *формулу Герона*: $\sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$,
1. где $p = (a + b + c)/2$ — *полупериметр*.
7. Поменять местами содержимое переменных A и B и вывести новые значения A и B.
8. Даны переменные A, B, C. Изменить их значения, переместив содержимое A в B, B — в C, C — в A, и вывести новые значения переменных A, B, C.
9. Даны переменные A, B, C. Изменить их значения, переместив содержимое A в C, C — в B, B — в A, и вывести новые значения переменных A, B, C.
10. Найти значение функции $y = 3 \cdot x^6 - 6 \cdot x^2 - 7$ при данном значении x.
11. Найти значение функции $y = 4 \cdot (x-3)^6 - 7 \cdot (x-3)^3 + 2$ при данном значении x.
12. Дано число A. Вычислить A^8, используя вспомогательную переменную и три операции умножения. Для этого последовательно находить A^2, A^4, A^8. Вывести все найденные степени числа A.
13. Дано число A. Вычислить A^{15}, используя две вспомогательные переменные и пять операций умножения. Для этого последовательно находить A^2, A^3, A^5, A^{10}, A^{15}. Вывести все найденные степени числа A.
14. Дано значение угла α в градусах ($0 < \alpha < 360$). Определить значение этого же угла в радианах, учитывая, что $180° = \pi$ радианов. В качестве значения π использовать `math.pi`.
15. Дано значение угла α в радианах ($0 < \alpha < 2 \cdot \pi$). Определить значение этого же угла в градусах, учитывая, что $180° = \pi$ радианов. В качестве значения π использовать `math.pi`.

16. Дано значение температуры T в градусах Фаренгейта. Определить значение этой же температуры в градусах Цельсия. Температура по Цельсию T_C и температура по Фаренгейту T_F связаны следующим соотношением:

$$T_C = (T_F - 32) \cdot 5/9.$$

17. Дано значение температуры T в градусах Цельсия. Определить значение этой же температуры в градусах Фаренгейта. Температура по Цельсию T_C и температура по Фаренгейту T_F связаны следующим соотношением:

$$T_C = (T_F - 32) \cdot 5/9.$$

18. Известно, что X кг конфет стоит A рублей. Определить, сколько стоит 1 кг и Y кг этих же конфет.

19. Известно, что X кг шоколадных конфет стоит A рублей, а Y кг ирисок стоит B рублей. Определить, сколько стоит 1 кг шоколадных конфет, 1 кг ирисок, а также во сколько раз шоколадные конфеты дороже ирисок.

20. Скорость лодки в стоячей воде V км/ч, скорость течения реки U км/ч ($U < V$). Время движения лодки по озеру T_1 ч, а по реке (против течения) — T_2 ч. Определить путь S, пройденный лодкой (путь = время · скорость). Учесть, что при движении против течения скорость лодки уменьшается на величину скорости течения.

21. Скорость первого автомобиля V_1 км/ч, второго — V_2 км/ч, расстояние между ними S км. Определить расстояние между ними через T часов, если автомобили удаляются друг от друга. Данное расстояние равно сумме начального расстояния и общего пути, проделанного автомобилями; общий путь = время · суммарная скорость.

22. Скорость первого автомобиля V_1 км/ч, второго — V_2 км/ч, расстояние между ними S км. Определить расстояние между ними через T часов, если автомобили первоначально движутся навстречу друг другу. Данное расстояние равно модулю разности начального расстояния и общего пути, проделанного автомобилями; общий путь = время · суммарная скорость.

23. Решить линейное уравнение $A \cdot x + B = 0$, заданное своими коэффициентами A и B (коэффициент A не равен 0).

24. Найти корни *квадратного уравнения* $A \cdot x^2 + B \cdot x + C = 0$, заданного своими коэффициентами A, B, C (коэффициент A не равен 0), если известно, что дискриминант уравнения положителен. Вывести вначале меньший, а затем больший из найденных корней. Корни квадратного уравнения находятся по формуле $x_{1,2} = (-B \pm \sqrt{D})/(2 \cdot A)$, где D — *дискриминант*, равный $B^2 - 4 \cdot A \cdot C$.

25. Дано двузначное число. Найти сумму и произведение его цифр.

26. Дано двузначное число. Вывести число, полученное при перестановке цифр исходного числа.
27. Дано трехзначное число. Используя одну операцию деления нацело, вывести первую цифру данного числа (сотни).
28. Дано трехзначное число. Вывести вначале его последнюю цифру (единицы), а затем — его среднюю цифру (десятки).
29. Дано трехзначное число. Найти сумму и произведение его цифр.
30. Дано трехзначное число. Вывести число, полученное при прочтении исходного числа справа налево.
31. Дано трехзначное число. В нем зачеркнули первую слева цифру и приписали ее справа. Вывести полученное число.
32. Дано трехзначное число. В нем зачеркнули первую справа цифру и приписали ее слева. Вывести полученное число. 15
33. Дано трехзначное число. Вывести число, полученное при перестановке цифр сотен и десятков исходного числа (например, 123 перейдет в 213).
34. Дано трехзначное число. Вывести число, полученное при перестановке цифр десятков и единиц исходного числа (например, 123 перейдет в 132).
35. Дано целое число, большее 999. Используя одну операцию деления нацело и одну операцию взятия остатка от деления, найти цифру, соответствующую разряду сотен в записи этого числа.
36. Дано целое число, большее 999. Используя одну операцию деления нацело и одну операцию взятия остатка от деления, найти цифру, соответствующую разряду тысяч в записи этого числа.
37. С начала суток прошло N секунд (N — целое). Найти количество полных минут, прошедших с начала суток.
38. С начала суток прошло N секунд (N — целое). Найти количество полных часов, прошедших с начала суток.
39. С начала суток прошло N секунд (N — целое). Найти количество секунд, прошедших с начала последней минуты.
40. С начала суток прошло N секунд (N — целое). Найти количество секунд, прошедших с начала последнего часа.
41. С начала суток прошло N секунд (N — целое). Найти количество полных минут, прошедших с начала последнего часа.
42. Дни недели пронумерованы следующим образом: 0 — воскресенье, 1 — понедельник, 2 — вторник, …, 6 — суббота. Дано целое число K, лежащее в диапазоне 1–365. Определить номер дня недели для K-го дня года, если известно, что в этом году 1 января было понедельником.

43. Дни недели пронумерованы следующим образом: 0 — воскресенье, 1 — понедельник, 2 — вторник, …, 6 — суббота. Дано целое число *K*, лежащее в диапазоне 1–365. Определить номер дня недели для *K*-го дня года, если известно, что в этом году 1 января было четвергом.
44. Дни недели пронумерованы следующим образом: 1 — понедельник, 2 — вторник, …, 6 — суббота, 7 — воскресенье. Дано целое число *K*, лежащее в диапазоне 1–365. Определить номер дня недели для *K*-го дня года, если известно, что в этом году 1 января было вторником.
45. Дни недели пронумерованы следующим образом: 1 — понедельник, 2 — вторник, …, 6 — суббота, 7 — воскресенье. Дано целое число *K*, лежащее в диапазоне 1–365. Определить номер дня недели для *K*-го дня года, если известно, что в этом году 1 января было субботой. 16

Литература и источники в Интернет

1. Numeric and Mathematical Modules [электронный ресурс]: https://docs.python.org/3/library/numeric.html (дата обращения 07.08.2017).