Введение в типы

Встроенные типы объектов

Тип объекта	Пример литерала/создания
Числа	`1234, 3.1415, 3+4j, 0b111, Decimal(), Fraction()`
Строки	`'spam', "Bob's", b'a\x01c', u'sp\xc4m'`
Списки	`[1, [2, 'three'], 4.5], list(range(10))`
Словари	`{'food': 'spam', 'taste': 'yum'}, dict(hours=10)`
Кортежи	`(1, 'spam', 4, 'U'), tuple('spam'), namedtuple`
Файлы	`open('eggs.txt'), open(r'C:\ham.bin', 'wb')`
Множества	`set('abc'), {'a', 'b', 'c'}`
Прочие основные типы	Булевские значения, сами типы, `None`
Типы программных единиц	Функции, модули, классы (часть IV, часть V, часть VI)
Типы, связанные с реализацией	Скомпилированный код, трассировки стека (часть IV, часть VII)

Числа / Модули для работы с числами

```
>>> 123 + 222 # Целочисленное сложение
345
>>> 1.5 * 4 # Умножение с плавающей точкой
6.0
>>> 2 ** 100 # 2 в степени 100
1267650600228229401496703205376
>>> import math
>>> math.pi
3.141592653589793
>>> math.sqrt(85)
9.219544457292887
>>> import random
>>> random.random()
0.7082048489415967
>>> random.choice([1, 2, 3, 4])
1
```

Форматирование значений

Python предлагает сразу несколько способов выполнения форматирования значений, включая использование форматирующего оператора (%), функцию и метод format, шаблонные строки и f-строки

```
print("%.2f" % x)
print("%s съел %d пирожков!" % (name, numCookies))
```

Форматирование значений

Фрагмент кода:
Результат:
Описание:
	`"%d" % x`
	`"12"`
	Значение из переменной x отображается в формате целого числа

Фрагмент кода:
Результат:
Описание:
	`"%f" % y`
	`"-2.75"`
	Значение из переменной y отображается в формате числа с плавающей запятой

Фрагмент кода:
Результат:
Описание:
	`"%d и %f" % (x, y)`
	`"12 и -2.75"`
	Значение из переменной x отображается в формате целого числа, а значение из переменной y отображается в формате числа с плавающей запятой. Остальные символы в строке остаются без изменений

Фрагмент кода:
Результат:
Описание:
	`"%.4f" % x`
	`"12.0000"`
	Значение из переменной x отображается в формате числа с плавающей запятой с четырьмя десятичными знаками

Форматирование значений

Фрагмент кода:
Результат:
Описание:
	`"%.1f" % y`
	`"-2.8"`
	Значение из переменной y отображается в формате числа с плавающей запятой с одним десятичным знаком. При выводе значение будет округлено, поскольку изначально в переменной находится значение с большим количеством десятичных знаков

Фрагмент кода:
Результат:
Описание:
	`"%10s" % z`
	`"    Andrew"`
	Значение из переменной z отображается в формате строки, занимающей минимум десять знакомест. Поскольку в слове Andrew шесть букв, к результату добавится четыре ведущих пробела

Фрагмент кода:
Результат:
Описание:
	`"%4s" % z`
	`"Andrew"`
	Значение из переменной z отображается в формате строки, занимающей минимум четыре знакоместа. Поскольку в слове Andrew шесть букв, результат окажется равен исходному значению переменной z

Фрагмент кода:
Результат:
Описание:
	`"%8i%8i" % (x, y)`
	`"      12      -2"`
	Значения из переменных x и y отображаются в формате целого числа, занимающего минимум восемь знакомест. При этом были добавлены ведущие пробелы. При преобразовании значения переменной y в целочисленный тип его дробная часть была отброшена (а не округлена)

Строки

Строки применяются для записи текстовой информации и произвольных совокупностей байтов (наподобие содержимого файла изображения). Они, так же, являются первым примером того, что в Python называется *последовательностью* — позиционно упорядоченной коллекцией других объектов. Для содержащихся элементов последовательности поддерживают порядок слева направо: элементы сохраняются и извлекаются по своим относительным позициям.

Операции над последовательностями

```
>>> S = 'Spam'   # Создать 4-символьную строку и присвоить ее некоторому имени
>>> len(S)       # Длина
4
>>> S[0]         # Первый элемент в S, который индексируется по позиции,
                 # начиная с нуля
'S'
>>> S[1]         # Второй элемент слева
'p'

>>> S[-1]          # Последний элемент с конца в S
'm'
>>> S[-2]          # Второй элемент с конца в S
'a'

>>> S[-1]          # Последний элемент с конца в S
'm'
>>> S[-2]          # Второй элемент с конца в S
'a'
```

Операции над последовательностями

```
>>> S[1:]          # Все после первого элемента (1:len(S))
'pam'
>>> S              # Сама строка S не изменилась
'Spam'
>>> S[0:3]         # Все кроме последнего элемента
'Spa'
>>> S[:3]          # То же, что и S[0:3]
'Spa'
>>> S[:-1]         # Снова все кроме последнего элемента, но проще (0:-1)
'Spa'
>>> S[:]           # Вся строка S как копия верхнего уровня (0:len(S))
'Spam'
>>> S
'Spam'
>>> S + 'xyz'      # Конкатенация
'Spamxyz'
>>> S              # S не изменяется
'Spam'
>>> S * 8          # Повторение
'SpamSpamSpamSpamSpamSpamSpamSpam'
```

Неизменяемость (immutability)

```
>>> S
'Spam'
>>> S[0] = 'z'       # Неизменяемые объекты модифицировать нельзя
...текст сообщения об ошибке не показан...
TypeError: 'str' object does not support item assignment
Ошибка типа: объект str не поддерживает присваивание в отношении элементов
>>> S = 'z' + S[1:]  # Но мы можем выполнять выражения для создания новых объектов
>>> S
'zpam'
```

```
>>> S = 'shrubbery'
>>> L = list(S)                 # Развернуть в список: [...]
>>> L
['s', 'h', 'r', 'u', 'b', 'b', 'e', 'r', 'y']
>>> L[1] = 'c'                  # Изменить на месте
>>> ''.join(L)                  # Объединить с пустым разделителем
'scrubbery'
```

```
>>> B = bytearray(b'spam')      # Гибрид байтов/списка
>>> B.extend(b'eggs')           # b необходимо в Python 3.X, но не в Python 2.X
>>> B                           # B[i] = ord(x) тоже здесь работает
bytearray(b'spameggs')
>>> B.decode()                  # Преобразовать в обычную строку
'spameggs'
```

Методы, специфичные для строк

```
>>> S = 'Spam'
>>> S.find('pa')              # Найти смещение подстроки в S
1
>>> S
'Spam'
>>> S.replace('pa', 'XYZ')    # Заменить вхождения подстроки в S другой подстрокой
'SXYZm'
>>> S
'Spam'

>>> line = 'aaa,bbb,ccccc,dd'
>>> line.split(',')           # Разбить по разделителю в список подстрок
['aaa', 'bbb', 'ccccc', 'dd']

>>> S = 'spam'
>>> S.upper()                 # Преобразовать в верхний и нижний регистры
'SPAM'
>>> S.isalpha()               # Проверить содержимое: isalpha, isdigit и т.д.
True

>>> line = 'aaa,bbb,ccccc,dd\n'
>>> line.rstrip()             # Удалить пробельные символы с правой стороны
'aaa,bbb,ccccc,dd'
>>> line.rstrip().split(',')  # Скомбинировать две операции
['aaa', 'bbb', 'ccccc', 'dd']
```

Методы, специфичные для строк

```
>>> '%s, eggs, and %s' % ('spam', 'SPAM!')          # Выражение форматирования
                                                    # (все версии)
'spam, eggs, and SPAM!'
>>> '{0}, eggs, and {1}'.format('spam', 'SPAM!')    # Метод форматирования
                                                    # (2.6+, 3.0+)
'spam, eggs, and SPAM!'
>>> '{}, eggs, and {}'.format('spam', 'SPAM!')      # Номера необязательны
                                                    # (2.7+, 3.1+)
'spam, eggs, and SPAM!'
```

```
>>> '{:,.2f}'.format(296999.2567)       # Разделители, десятичные цифры
'296,999.26'
>>> '%.2f | %+05d' % (3.14159, -42)     # Цифры, дополнение, знаки
'3.14 | -0042'
```

Чтение ввода

```
a = input()

name = input("Введите имя: ")

quantity = int(input("Сколько товаров вы желаете приобрести? "))

price = float(input("Какова цена за единицу товара? "))
```

Вывод результата

- print(1)
- print("Hello!")
- print(x)
- print("Когда x равен", x, ", y будет равен", y)
- print("Если умножить", x, "на", y, "получится", x * y)