Лекция №3

Основы программирования на Python

- Операторы Python
- Простые типы данных
- Операторы условия
- Циклы

Арифметические операторы

Оператор	Название	Объяснение	Примеры
+	Сложение	Суммирует два объекта	3 + 5 даст 8; 'a' + 'b' даст 'ab'
-	Вычитание	Даёт разность двух чисел; если первый операнд отсутствует, он считается равным нулю	-5.2 даст отрицательное число, а 50 - 24 даст 26.
*	Умножение	Даёт произведение двух чисел или возвращает строку, повторённую заданное число раз.	2 * 3 даст 6. 'la' * 3 даст 'lalala'.
**	Возведение в степень	Возвращает число x, возведённое в степень y	3 ** 4 даст 81 (т.е. 3 * 3 * 3 * 3)
/	Деление	Возвращает частное от деления x на y	4 / 3 даст 1.3333333333333333.
//	Целочисленное деление	Возвращает неполное частное от деления	4 // 3 даст 1.
%	Деление по модулю	Возвращает остаток от деления	8 % 3 даст 2. -25.5 % 2.25 даст1.5.

Побитовые операторы

Оператор	Название	Объяснение	Примеры
<<	Сдвиг влево	Сдвигает биты числа влево на заданное количество позиций.	2 << 2 даст 8. В двоичном виде 2 - 10. Сдвиг влево на 2 бита даёт 1000, в десятичном виде 8.
>>	Сдвиг вправо	Сдвигает биты числа вправо на заданное число позиций.	11 >> 1 даст 5. В двоичном виде 11 - 1011, что будучи смещённым на 1 бит вправо, даёт 101, а это десятичное 5
&	Побитовое И	Побитовая операция И над числами	5 & 3 даёт 1.
|	Побитовое ИЛИ	Побитовая операция ИЛИ над числами	5 | 3 даёт 7
^	Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ	Побитовая операция ИСКЛЮЧАЮЩЕЕ ИЛИ	5 ^ 3 даёт 6
~	Побитовое НЕ	Побитовая операция НЕ для числа x соответствует -(x+1)	~5 даёт -6.

Операторы сравнения

Оператор	Название	Объяснение	Примеры
<	Меньше	Определяет, верно ли, что x меньше y. Все операторы сравнения возвращают True или False.	5 < 3 даст False,
			Можно составлять произвольные цепочки сравнений: 3 < 5 < 7 даёт True.
>	Больше	Определяет, верно ли, что x больше y	5 > 3 даёт True. Если оба операнда - числа, то оба преобразуются к одинаковому типу. Иначе оба возвращается False.
<=	Меньше равно	Определяет, верно ли, что x меньше или равно y	x = 3; y = 6; x <= y даёт True.
>=	Больше равно	Определяет, верно ли, что x больше или равно y	x = 4; y = 3; x >= 3 даёт True.
==	Равно	Равны ли объекты	x = 2; y = 2; x == y даёт True.x = 'str'; y = 'stR'; x == y даёт False.

Логические операторы

Оператор	Название	Объяснение	Примеры
not	Логическое НЕ	Если x == True, вернётся False. Если x равно False, получим True.	x = True; not x даёт False.
and	Логическое И	x and y даёт False, если x равно False , в противном случае возвращает значение y	x = 0; y = 1; x and y возвращает 0, поскольку x равно 0.
or	Логическое ИЛИ	Если x равно True, в результате получим True, в противном случае получим значение y	x = True; y = False; x or y даёт True. Здесь также может производиться укороченная оценка выражений.

Операторы идентичности и включения

Оператор	Название	Объяснение	Примеры
is	Идентичность	x is y даёт True, если x и y - это один и тот же объект, а не просто имеют одинаковые значения	x = y = [1,2,3]; x is y вернет True, при x = [1,2,3], y = [1,2,3] будет False.
is not	Неидентичность	x is not y даёт False, если x и y - это один и тот же объект.	x = y = [1,2,3]; x is not y вернет False, при x = [1,2,3], y = [1,2,3] будет True.
in	Проверяет наличие	x in y даёт True, если x присутствует внутри y	x = 1; y = [1,2,3,4,5,6,7,8,9,10]; x in y вернет True
not in	Проверяет отсутствие	x not in y даёт True, если x отсутствует внутри y	x = 1; y = [1,2,3,4,5,6,7,8,9,10]; x not in y вернет False

Типы данных

В Python всего 2 простых типа данных:

- число (256),
- строка (“256”).

Остальные типы: кортеж, список, множество, словарь, - являются составными, т.е. представляют собой набор значений простых типов, упорядоченных определенным образом.

Также Python включает в себя специальные типы: None (для обозначения неинициализированных переменных), NotImplemented (для информирования об определенных ошибках) и Ellipsis (то же, что ‘…’ - в срезах массивов, в качестве заглушек в функциях).

Еще есть тип File для работы с файлами и все пользовательские типы, обозначаемые ключевым словом Class – классы.

Число (Number) - Immutable

- Обычные целые числа: int('5'), 1, 5
- Восьмеричные числа(тоже int): 020
- Шестнадцатеричные числа: 0xA, 0xa
- Числа с плавающей точкой: float('1.4'), 1.4, 1e7
- Комплексные числа: complex(1, 2), 5j, 5J, 3+4j, z.real, z.imag
- Логический (булев) тип: True, False

Логический тип в Python не выделяется в отдельный тип, т.к. считается одним из представлений типа Число. Логические выражения, имеющие значение True, считаются равными 1. Логические выражения, имеющие значение False, считаются равными 0.

Ноль, пустая строка (‘’), специальное значение ‘None’ считаются Ложью (False). Все остальное – Истина (True).

Функции для работы с числами

С числами связаны следующие операции, кроме перечисленных ранее:

int(x [,base]) - преобразует x к целочисленному типу, base - основание системы счисления, если x - строка.

float(x) - преобразует x к числу с плавающей точкой (вещественному числу).

complex(real [,imag]) – создает комплексное число

ord(x) – преобразует отдельный символ к целому числу согласно таблице ASCII.

hex(x) – преобразует число к 16-ричному виду в строковом представлении

oct(x) – преобразует число к 8-ричному виду в строковом представлении

round(n[, ndigits]) – округляет вещественное число до указанного разряда после запятой (по умолчанию до нулевого, т.е. до целой части).

Примеры работы с числами

Примеры для практики:

```
>>> print(2 ** 150)
1427247692705959881058285969449495136382746624
>>> print(int(5.89), round(5.89), round(5.89, 1))
5 6 5.9
>>> import math
>>> math.pi
3.1415926535897931
>>> print(math.sin(math.pi/2))
1.0
```

Строка (String) - Immutable

Фактически строка состоит из отдельных символов, но в Python нет понятия символа (это тоже строка только из одного элемента).

Доступ к элементам строки или подстрокам осуществляется по индексам или путем указания среза:

```
s1 = 'Hello World!'
s2 = "Python Programming!"
print("s1[0]: ", s1[0])
print("s2[1:5]: ", s2[1:5])

s1[0]: H
s2[1:5]: ytho
```

Строка является неизменяемой (immutable) последовательностью. Т.е. нельзя, к примеру, заменить какой-либо элемент в строке:

```
>>> s1[1] = 'a'
Traceback (most recent call last):
    File "<input>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Работа со строками

Зато можно создать новую строку с использованием старой:

```
>>> s3 = s1[0] + 'a' + s1[2:len(s1)]
>>> print(s3)
Hallo World
```

Обычные строки хранятся из расчета 8 бит на символ. Это достаточно для кодирования всех символов таблицы ASCII. Для использования спецсимволов и различных языков мира это недостаточно. Для этого были добавлены unicode строки, в которых под символ отводится 16 бит.
Unicode строка задается следующим образом:

```
us = u'Hello World'
```

Преобразование обычной строки в Unicode строку выполняется так:

```
us1 = unicode(s1)
```

Преобразование из другого типа в строку выполняется так:

```
>>> print(str(us1))
Hello World
>>> print(str(5)+str(2))
52
```

Строковые операторы

```
a = 'Hello'
b = 'Python'
```

Оператор	Описание	Пример
+	Конкатенация. Соединяет две строки в третью	a + b даст HelloPython
*	Повторение. Новая строка получается повторением исходной заданное количество раз.	a*2 даст HelloHello
[]	Индекс. Возвращает символ строки по индексу (начиная с 0).	a[1] даст e
[:]	Срез. Позволяет получить подстроку с начального индекса по конечный (можно указать шаг).	a[1:4] даст ell
in	Включение. Возвращает Истину, если указанный символ присутствует в строке.	H in a даст 1
not in	Не включение. Возвращает истину, если указанный символ отсутствует в строке.	M not in a даст 1
r/R	«Сырая» строка. Подавляет Escape-символы. Обозначается добавлением r или R перед открывающейся кавычкой.	print r'\n' выводит \n and print R'\n' выводит \n
%	Форматирование. Выполняет форматирование строки.	

Встроенные методы строк и функции для работы со строками

Методы и функции	Описание
.capitalize()	Делает заглавной первую букву строки.
.center(width, fillchar)	Центрирует строку по указанной ширине, заполняя оставшиеся места пробелами (по умолчанию), либо символом fillchar.
.count(str, beg= 0,end=len(string))	Считает, сколько раз подстрока str встречается в строке, начиная с индекса beg по индекс end.
.decode(encoding='UTF-8',errors='strict')	Преобразует строку из указанной кодировки в кодировку по умолчанию.
.encode(encoding='UTF-8',errors='strict')	Преобразует строку из кодировки по умолчанию в указанную кодировку.
.endswith(suffix, beg=0, end=len(string))	Определяет заканчивается ли строка (или подстрока, если заданы beg и end) заданным суффиксом.
.expandtabs(tabsize=8)	Заменяет символы табуляции на указанное число пробелов.
.find(str, beg=0 end=len(string))	Определяет, встречается ли подстрока str в строке (или подстроке, если заданы beg и end); возвращает начальный индекс найденной подстроки, либо -1, если ничего не найдено.
.index(str, beg=0, end=len(string))	Так же как find(), но выбрасывает исключение, если str не найдена.
.isalnum()	Возвращает True, если строка содержит по крайней мере один символ, и все символы в строке - либо цифры, либо буквы. False в противном случае.

Встроенные методы строк и функции для работы со строками

Методы и функции	Описание
.isalpha()	Возвращает True, если строка содержит по крайней мере один символ, и все символы в строке - буквы. False в противном случае.
.isdigit()	Возвращает True, если все символы в строке - цифры. False в противном случае.
.islower()	Возвращает True, если строка содержит по крайней мере один символ, и все символы в строке строчные (в нижнем регистре). False в противном случае.
.isnumeric()	Возвращает True, если все символы в unicode-строке - численные. False в противном случае.
.isspace()	Возвращает True, если строка содержит только пробелы. False в противном случае.
.istitle()	Возвращает True, если все слова в строке начинаются с заглавной буквы. False в противном случае.
.isupper()	Возвращает True, если строка содержит по крайней мере один символ, который можно привести к верхнему и нижнему регистру, и все символы в строке прописные (в верхнем регистре). False в противном случае.
.join(seq)	Формирует строку из элементов последовательности seq, разделителем является строка, чей метод вызван.
len(string)	Возвращает длину строки.
.ljust(width[, fillchar])	Выравнивает строку по левому краю, до ширины width заполняя оставшиеся места пробелами (по умолчанию), либо символом fillchar.
.lower()	Преобразует все прописные буквы в строке в строчные.
.lstrip()	Удаляет все пробелы из начала строки.
.maketrans()	Возвращает таблицу преобразования для использования в функции translate преобразования.
max(string)	Возвращает максимальный по алфавитному порядку символ в строке.

Встроенные методы строк и функции для работы со строками

Методы и функции	Описание
min(string)	Возвращает минимальный по алфавитному порядку символ в строке.
.replace(old, new [, max])	Замещает все (либо не больше max, если max задан) подстроки old в строке на подстроку new.
.rfind(str, beg=0,end=len(string))	Так же как find(), только при поиске проходит строку в обратном направлении.
.rindex(str, beg=0, end=len(string))	Также как index(), только при поиске проходит строку в обратном направлении.
.rjust(width,[, fillchar])	Выравнивает строку по правому краю, до ширины width заполняя оставшиеся места пробелами (по умолчанию), либо символом fillchar.
.rstrip()	Удаляет все пробелы в конце строки.
.split(str="", num=string.count(str))	Разделяет строку по символу str (по умолчанию, пробел) и возвращает список подстрок. Если указан num, то разделяет не больше, чем на num подстрок.
.splitlines(num=string.count('\n'))	Разделяет строку по символу перевода строки (\n) и возвращает список подстрок (уже без \n). Если указан num, то разделяет не больше, чем на num подстрок.
.startswith(str, beg=0,end=len(string))	Определяет начинается ли строка (или подстрока, если заданы beg и end) с подстроки str.
.strip([chars])	Выполняет сразу и lstrip(), и rstrip() над строкой.

Встроенные методы строк и функции для работы со строками

Методы и функции	Описание
.swapcase()	Инвертирует регистр для всех букв в строке.
.title()	Возвращает строку, в которой все слова начинаются с заглавной буквы.
.translate(table, deletechars="")	Преобразует строку в соответствии с таблицей преобразования, удаляя символы в списке deletechars.
.upper()	Преобразует строчные буквы в строке в заглавные.
.zfill(width)	Заполняет строку нулями с начала до ширины width.
.isdecimal()	Возвращает True, если все символы в unicode-строке - цифры. False в противном случае.

Оператор условия IF

```
if <логическое выражение>:
    <блок кода>
elif <логическое выражение>:
    <блок кода>
…
else:
    <блок кода, который должен выполниться, если остальные блоки не выполнялись>
```

```
if <логическое выражение>:
    <блок кода>
else:
    <блок кода, который должен выполниться, если остальные блоки не выполнялись>
```

```
if <логическое выражение>:
    <блок кода>
elif <логическое выражение>:
    <блок кода>
```

```
if <логическое выражение>:
    <блок кода>
```

Оператор условия IF

```
number = 23
guess = int(input('Введите целое число : '))

if guess == number:
    print('Поздравляю, вы угадали') # Здесь начинается и заканчивается новый блок
elif guess < number:
    print('Нет, загаданное число немного больше этого.') # Ещё один блок
else:
    print('Нет, загаданное число немного меньше этого.')
    # чтобы попасть в else, guess должно быть больше, чем number
print('Завершено')
# Это последнее выражение выполняется всегда после выполнения оператора if

$ python3 if.py
Введите целое число : 50
Нет, загаданное число немного меньше этого.
Завершено
$ python3 if.py
Введите целое число : 22
Нет, загаданное число немного больше этого.
Завершено
$ python3 if.py
Введите целое число : 23
Поздравляю, вы угадали
Завершено
```

Оператор цикла WHILE

```
while <логическое выражение>:
    <тело цикла>
else <логическое выражение>:
    <блок кода, который должен выполниться, если тело цикла не выполнилось>
```

```
while <логическое выражение>:
    <тело цикла>
```

```
>>> while i != 5:
...     i += 1
...
>>> print(i)
5

>>> i = 0
>>> while i != 5:
...     i += 1
...     if i == 3:
...         break
...
>>> print(i)
3
```

Оператор цикла WHILE

```
>>> i = 0
>>> while i != 5:
...     i += 1
... else:
...     print('Цикл дошел до конца!')
...
Цикл дошел до конца!
>>> print(i)
5

>>> i = 0
>>> while i != 5:
...     i += 1
...     if i % 2:
...         print(i)
...     else:
...         continue
...
1
3
5
>>> print(i)
5
```

Оператор цикла FOR

```
for <элемент> in <последовательность>:
    <тело цикла>
```

```
>>> for i in range(1, 5):
...     print(i)
...
1
2
3
4

>>> l = [1,2,3,4]
>>> for i in l:
...     print(i)
...
1
2
3
4

>>> for i in [1,2,3,4]:
...     print(i * 2)
...
2
4
6
8
```