Голодный питон

Питон это мультипарадигмальный язык, то есть он: объектно-ориентированный, рефлективный, императивный, функциональный, аспектно-ориентированный, со строгой динамической типизацией. . .

Python — красивый и местами загадочный язык. И даже зная его весьма неплохо, рано или поздно находишь для себя нечто такое, что раньше не использовал.

Петербург
2016

Среда

Официальный сайт питона: http://python.org

Свежие дистрибутивы можно скачать здесь:

Python 2.7.x

https://www.python.org/ftp/python/2.7.11/python-2.7.11.msi

Python 3.x

https://www.python.org/ftp/python/3.5.1/python-3.5.1.exe

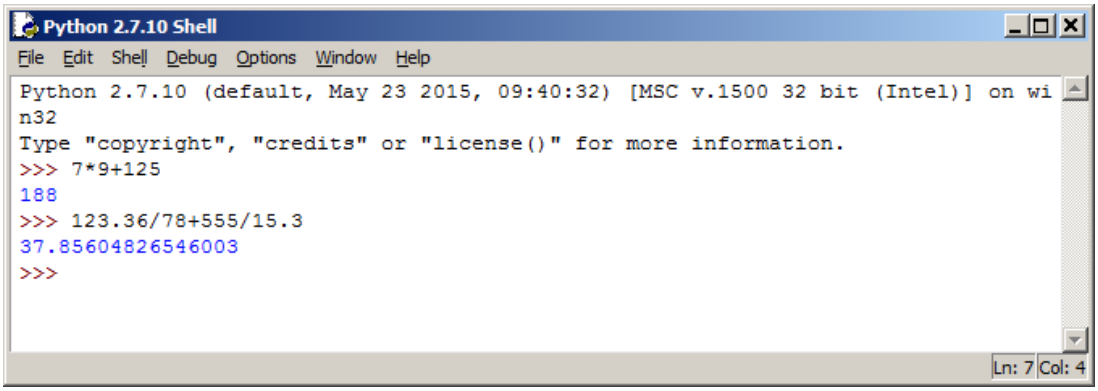
Это не предыдущая и последующая версия, а два, немного разных языка.

Считается, что Python 3.x - правильный, а Python 2.7.x - удобный.

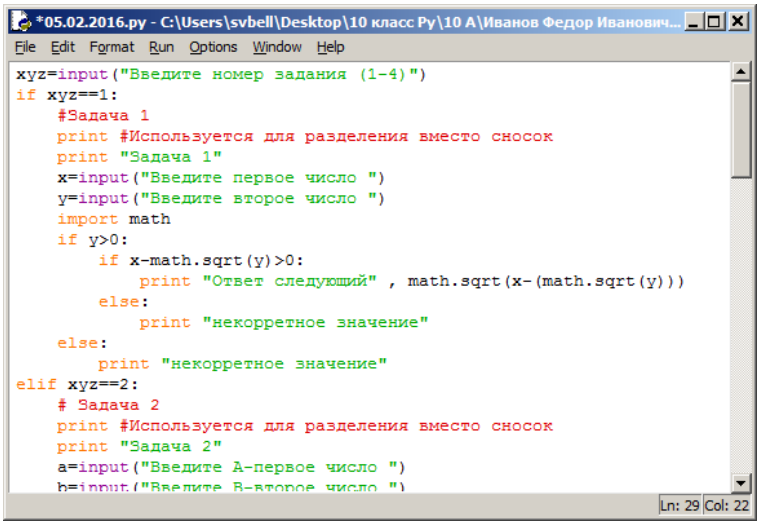
Интерактивная среда питона: https://trinket.io/python/41462f0f16

Среда разработки идущая в дистрибутиве называется: IDLE.

При запуске открывается окно **Shell**. Это удобный интерактивный режим для тестирования различных возможностей языка. Интерпретатор выполняет команды построчно: пишешь строку, нажимаешь Enter, интерпретатор выполняет ее, наблюдаешь результат. В этом режиме ключевое слово **print** писать не обязательно, и можно использовать Питон как калькулятор.



Окно для работы с программами вызывается пунктом меню **File - New File**



Запуск программы осуществляется пунктом меню **Run - Run Module** либо F5. Если текст программы не сохранен, то среда предлагает сохранить его. Без сохранения программа на исполнение не запускается.

Хотя **Shell** и позволяет перемещаться по ранее набранным строкам, вызвать на обработку можно только последнюю строчку.

В **Shell** высветить ранее набранные строчки можно комбинацией Alt + p

Элементы языка (Операторы, функции, объекты)

Операторы

Внешний вид операторов абсолютно индивидуален. Каждый из них изучается отдельно.

Слова операторов это ключевые слова – (их нельзя употреблять в качестве идентификаторов)

['**and**', '**as**', '**assert**', '**break**', '**class**', '**continue**', '**def**', '**del**', '**elif**', '**else**', '**except**', '**exec**', '**finally**', '**for**', '**from**', '**global**', '**if**', '**import**', '**in**', '**is**', '**lambda**', '**not**', '**or**', '**pass**', '**print**', '**raise**', '**return**', '**try**', '**while**', '**with**', '**yield**']

Список ключевых слов можно получить следующим заклинанием в **Shell**:

import keyword; **print** keyword.kwlist

Заметим, что все ключевые слова записываются маленькими буквами.

Операторы размещается каждый на отдельной строке.

Можно, конечно, разместить на одной строке несколько операторов через точку с запятой, но этого делать не стоит. Можно, также, разместить один оператор на нескольких строчках при помощи обратного слэша, но этого делать, также, не стоит.

Функции

Функции используют либо в выражениях, либо в операторах присваивания, либо просто так. Внешне они выглядят так: имя функции, список формальных параметров в скобках. Даже если параметров нет, скобки обязательны.

У Питона большое количество встроенных функций. Кроме того, есть масса внешних модулей, которые легко подключить, и работать с функциями, содержащимися там.

Все функции возвращают какое-нибудь значение, но использовать его не обязательно. Если нет желания использовать это значение, то и присваивание делать необязательно. В этом случае использование функции ничем не отличается от того, что на других языках зовется подпрограммой. Заметим, что подавляющее большинство функций, как встроенных, так и подключаемых, записываются маленькими буквами.

Объекты: (Переменные, файлы, и пр.)

Идентификаторы объектов: Слова, начинающиеся с буквы. Ключевые слова употреблять нельзя. Размер играет роль.

Поскольку питон - объектно-ориентированный язык, то у объектов (в т.ч. и переменных, есть не только имя, тип, значение, но и методы.

Метод это такая функция, которая возникает только с объектом. Внешне они выглядят так: имя объекта, точка, имя метода, список формальных параметров в скобках. Даже если параметров нет, скобки обязательны.

Типы объектов.

Описание объектов осуществляется через инициацию (присваивание). Что присвоили, такой и тип. Одной и той же переменной можно присваивать в разных местах программы разные типы. Ключевой символ присваивания - "=".

Типы бывают:

Целое число (**int**)	a=50
Длинное целое число (**long**)	a=50L
Вещественное число (**float**)	a=50. a=5e2
Строка (**str**)	a="Hello, Python!" a='Hello, Python!'
Список (**list**)	a=[1, 2., "mir", 'trud']

Строковые константы могут быть обрамлены как ординарными, так и двойными кавычками. Это сделано для того, чтобы внутри можно было использовать разные кавычки как символы.

Иногда может возникнуть необходимость преобразовать один тип данных в другой. Для этого существуют специальные встроенные функции Python.:

Пусть, x - вещественное число,

s - строка, изображающая целое число,

i - целое число

Функция	Описание	Пример
int(x)	Преобразовывает x в целое число.	**int**(12.4) $\rightarrow$ 12
int(s)	Преобразовывает s в целое число.	**int**("12") $\rightarrow$ 12

int(s, i)	Преобразовывает s в целое десятичное число, читая, что s - это число в системе счисления i. i можно задать от 2 до 36	**int**("12", 4) → 6
long(x)	Преобразовывает x в **long**.	**long**(20) → 20L
float(x)	Преобразовывает x в число с плавающей точкой.	**float**(10) → 10.0
str(x)	Преобразовывает x в строку.	**str**(10) → '10'
list(s)	Преобразовывает s в список.	**list**("Python") → ["P","**y**","**t**","**h**","**o**","n"]

Числовые переменные

Основные операций для чисел:

A + B - сумма;

A - B - разность;

A * B - произведение;

A / B - частное;

A // B - целая часть от частного;

A ** B - возведение в степень. Полезно помнить, что квадратный корень из числа x - это x ** 0.5, а корень степени n это x ** (1 / n).

A % B - остаток от деления A на B

Питон поддерживает длинную арифметику:

3 ** 120

179701029991443121041317982950960503973147562753785110 6401L

Правила определения приоритетов операций такие:

1. Выполняются возведения в степень **справа налево**, то есть `3**3**3` это 3^9 а не 9^3.
2. Выполняются унарные минусы (отрицания).

   Если унарные операторы расположены **слева** от оператора **, то возведение в степень имеет **больший** приоритет, а если справа - то меньший

   -2**-2 это -(2 ** (-2)), т.е. -0,25
3. Выполняются умножения и деления слева направо. Операции умножения и деления имеют одинаковый приоритет , `-b/2*a` это не $\frac{-b}{2a}$.

4. Выполняются сложения и вычитания слева направо. Операции сложения и вычитания имеют одинаковый приоритет.
5. Если операнды целые, то и результат целый. Если хоть один из операндов вещественный, то и результат вещественный. Исключение - деление двух целых чисел в Python 3.x результат вещественный

Важные встроенные функции числовых переменных.

abs(x) - Возвращает абсолютную величину (модуль числа).

bin(x) - Преобразование целого числа в двоичную строку.

oct(x) - Преобразование целого числа в восьмеричную строку.

hex(x) - Преобразование целого числа в шестнадцатеричную строку.

pow(x, y[, r]) - (x ** y) % r.

round(X [, N]) - Округление до N знаков после запятой.

Подключаемые модули.

Модуль **math** – один из наиважнейших в Python. Этот модуль предоставляет обширный функционал для работы с числами.

Подключение модуля : **import math**

Модуль **random** предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности.

Подключение модуля : **import random**

Строки

Основные операции над строками:

A + B — конкатенация;

A * N — повторение N раз, значение N должно быть целого типа.

Возможны разные комбинации этих операций со скобками. Например

2*(' '+'бар'*2+'ис')+ 'ович' - барбарис барбарисович

В справочной литературе употребляется таинственный термин «Срез (slice)»,

В случае строк, это не что иное, как подстрока.

Примеры срезов:

Пусть s='Съешь еще этих мягких французских булок'

Срез	Значение
S	Съешь еще этих мягких французских булок
s[0]	С

s[1]	ъ
s[2:4]	еш
s[:4]	Съеш
s[-1]	к
s[-2]	о
s[-5:]	булок

Пояснение:

s[n,m] – подстрока строки s начиная с символа n до символа, **предшествующего** m.

Обратите внимание, что верхняя граница диапазона индексов не включительная!

Символы считаются от нуля.

Если n отсутствует, то срез начинается с начала строки,

Если m отсутствует, то срез продолжается до конца строки,

Если n или m отрицательные, то счет от конца строки.

В Питоне строки являются неизменяемыми, их невозможно изменить.

Можно лишь старой переменной присвоить целиком вновь сформированную строку.

Важные встроенные функции строк.

i=**int**(s) - Преобразование строки, которая выглядит как целое число, в целое число

s=**str**(x) - Преобразование числа в строку. Число может быть как целым, так и действительным.

l=**len**(s) - возвращает длину строки

Важнейшие методы строковой переменной:

S.**find**(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.**rfind**(str, [start],[end])	Поиск подстроки в строке с конца. Возвращает номер последнего вхождения или -1
S.**replace**(искомое, замена)	Замена искомого
S.**upper**()	Преобразование к верхнему регистру

S.**lower**()	Преобразование к нижнему регистру
S.**count**(str, [start],[end])	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)

Списки

В питоне, для работы со структурами данных, есть: списки, кортежи, наборы, словари, нечто, именуемое массивы, и т.п. Самое простое для понимания - это списки. Поэтому, в качестве массивов мы будем использовать списки (ими и ограничимся).

Список это индексированная последовательность значений, разделенных запятыми, заключенная в квадратные скобки.

Элементы списка не обязательно должны быть одного типа.

Как и для строк, для списков нумерация индексов начинается с нуля.

В отличие от строк, можно менять отдельные элементы списка и поддиапазоны.

Основные операции над списками:

A + B слияние двух списков;

A * N — повторение списка N раз, значение N должно быть целого типа.

Опять, таки, возможны разные комбинации этих операций со скобками. Например

a=3*([6]*2+[4])

[6, 6, 4, 6, 6, 4, 6, 6, 4]

Таинственный термин «Срез (slice)» в случае списков тоже имеет место быть. Только в этом случае, это не что иное, как поддиапазон.

В списке поддиапазон можно удалить оператором **del**:

a=[0, 1, 2, 3, 4, 5, 6, 7, 8]

del a[3:5]

[0, 1, 2, 5, 6, 7, 8]

при помощи оператора **in** можно определить, есть ли элемент в списке.

3 **in** a → **False**

При помощи среза осуществляется создание копии списка.

L1 = L2[:] - создание копии списка.

Важно!

Если мы запишем вот так:

L1 = L2

то произойдет создание второй ссылки на список, а не копии этого списка. Другими словами, создаются две ссылки на один и тот же объект, а не две копии. Все, что мы будем делать со списком L1, будет происходить и со списком L2. И наоборот.

Важнейшие встроенные функции списков:

len (list) - получение количества элементов списка.

max(list), **min**(list) - получение максимальное, минимальное значение

sum(list) - получение суммы списка

sorted(list) - получение отсортированного списка

set(list) - получение разных уникальных элементов списка

reserved(list) - получение списка в обратном порядке

map(func, list) - получение нового списка из заданного путем применения к каждому элементу заданного списка функции func.

range([start,] N, [step]) - получение списка чисел арифметической прогрессии, начиная от start заканчивая N-1 с шагом step. Если start отсутствует, то начиная от 0. Если step отсутствует, то с шагом 1.

Примеры:
range(9)
[0, 1, 2, 3, 4, 5, 6, 7, 8]
range(3,9)
[3, 4, 5, 6, 7, 8]
range(3,9,2)
[3, 5, 7]

Важнейшие методы списков:

list.**append**(x) - Добавляет элемент в конец списка

list.**extend**(L) - Расширяет список list, добавляя в конец все элементы списка L

list.**insert**(i, x) - Вставляет на i-ый элемент значение x

list.**remove**(x) - Удаляет первый элемент в списке, имеющий значение x

list.**index**(x, [start [, end]]) - Находит первый элемент от start до end равный x

list.**count**(x) - Возвращает количество элементов со значением x

list.**sort**([**key** = функция]) - Сортирует список на основе функции

list.**reverse**()- Разворачивает список

Нужно отметить, что методы списков, в отличие от строковых методов, изменяют сам список, а потому результат выполнения не нужно записывать в эту переменную.

Метод строки, который возвращает список:

S.**split**(разделитель) - Формируется список путем разбиения строки по разделителю. Если разделитель отсутствует, то строка разбивается по

пробелу. Причем, несколько подряд идущих пробелов считаются за один.

Будьте внимательны: если Вам надо отсортировать список, то применяйте метод list.**sort**(), а если вам нужно использовать отсортированный список, не портя изначальный, то функцию **sorted**(list) то же относится к методу **reverse** и функции **reserved**

Примеры работы со списком:

список из одного элемента	a = [7]	теперь список a выглядит так: [7]
добавляем к нему элемент 5	a.**append**(5)	теперь список a выглядит так: [7, 5]
добавляем к нему элемент 6	a = a + [6]	теперь список a выглядит так: [7, 5, 6]
список из трех элементов	b = [1, 2, 3]	теперь список b выглядит так: [1, 2, 3]
список из трех элементов	c = **range**(3)	теперь список c выглядит так: [0, 1, 2]
список из трех элементов	**d** = 3*[0]	теперь список **d** выглядит так: [0, 0, 0]
объединяем списки a и b	c = a + b	теперь список c выглядит так: [7, 5, 6, 1, 2, 3]
выведем первый элемент	**print** c[0]	выведет 7 - первый элемент списока (нумерация начинается с нуля)
выведем последний элемент списка	**print** c[-1]	выведет 3
меняем значение элемента списка с индексом один на 8	c[1]=8	теперь список c выглядит так: [7, 8, 5, 1, 2, 3]
удалим элемент списка, который равен 2	c.**remove**(2)	теперь список c выглядит так: [7, 8, 5, 1, 3]
удалим элемент списка с индексом два	**del** c[2]	теперь список c выглядит так: [7, 8, 1, 3]

Двумерные массивы

В качестве элементов списка может быть что угодно. Например, список. Таким образом, формируется понятие двумерного массива как списка списков.

Логические выражения и логический тип данных

Логические константы: **True** и **False** (редкий случай, когда служебные слова питона не только маленькими буквами)

Операции сравнения:

== Равны ли оба операнда?

!=, <> Не равны ли оба операнда?

< (>) Меньше (больше) ли значение левого операнда, чем значение правого?

<= (>=) Меньше или равно (больше или равно) значение левого операнда, чем значение правого?

Логические операции представлены ключевыми словами

not (не), **and** (и), **or** (или).

Об операторных скобках

В питоне, равно как и в других языках, операторы цикла, условные операторы состоят из двух частей: заголовка оператора и тела оператора, которое состоит программного кода, заключенного в операторные скобки. В некоторых языках, это служебные слова, в некоторых – специальные символы. В питоне в качестве операторных скобок служат отступы. Отступы можно формировать пробелами, можно табуляцией. Главное, чтобы они были одинаковые. Следовательно, текст программы на питоне нужно писать не абы как, а аккуратно отображая уровни вложенности отступами.

Условный оператор

```
if x>0:
    a=4
    b=7
elif x=0:
    a=5
    b=7
else:
    a=8
    b=9
```


Оператор Цикла

В питоне есть два оператора цикла **for** и **while**.

Цикл **for** не совсем то, что обычно считается циклом **for** в других языках.

Имитация заголовка цикла **for** аналогичного циклу **for** в других языках осуществляется при помощи следующего магического заклинания:

for i **in range**(1, N+1, k):

Это, на поверхностном уровне, это эквивалентно следующему оператору Basic

FOR i=1 TO N STEP k

На самом деле, оператор **for** осуществляет перебор элементов списка (в данном случае, списка чисел арифметической прогрессии)

Итерацию списков в питоне можно делать несколькими различными способами:

простая итерация списка:
for x **in** L:

уникальная итерация:
for x **in set**(L):

сортированная итерация:
for x **in sorted**(L):

итерация в обратном порядке:
for x **in reversed**(L):

Цикл **while** это канонический цикл **while**, абсолютно такой-же, как во всех языках.

```
a=3
while a<100:
    a=a+22
```

Ввод-вывод

Для вывода на экран используется ключевое слово **print**.

Python 2.7.x : **print** - это оператор, значит, то, что мы хотим вывести, записывается сразу после него через запятую. Если в конце мы поставим запятую, то следующий **print** выведет свои данные на той же строке

Python 3.x : **print** - это функция, значит, то, что мы хотим вывести, записывается сразу после него, через запятую, в скобках. Чтобы следующий **print** вывел свои данные на той же строке, надо добавить end = ""

Для ввода данных с клавиатуры используется функция **input**([сообщение])

Когда выполняющаяся программа предлагает пользователю что-либо ввести, то пользователь может не понять, что от него хотят. С этой целью функция **input**() имеет необязательный аргумент-сообщение строкового типа; это сообщение будет появляться на экране и информировать человека о запрашиваемых данных.

Python 2.7.x : функция **input**() возвращает число. Для того, чтобы ввести строку, используется похожая функция **raw_input**()

Python 3.x : функция **input**() возвращает строку. Для того, чтобы получить число, эту строку надо преобразовать с помощью функции **int**() или **float**().

Что делать, если хочется, чтобы пользователь смог, например, ввести два числа через пробел? Это можно сделать, например, при помощи следующего магического заклинания:

В Python 2.7.x :

[x,y]=**map**(**int**, **raw_input**("x u y ").**split**())

В Python 3.x :

[x,y]=**map**(**int**, **input**("x u y ").**split**())

Разоблачение магии:
функция **raw_input**("Введите x и **y** через пробел ")
...нам приносит строчку, в которой находятся изображения двух чисел, разделенных пробелом...
метод .**split**()
...преобразует эту строчку в список, состоящий из двух строковых элементов с цифрами...
функция **map**(**int**, ...
... применяет ко всему этому списку функцию **int**, и таким образом получается список из двух чисел...
[x,**y**]=
...список их двух переменных ссылается на этот получившийся список, и, таким образом, в переменной x оказывается первое значение, а в переменной **y** второе...
Чисто научный опыт, как нельзя лучше доказывающий, что никаких чудес и магии не существует.

Файловый ввод-вывод

Файл это объект типа файл. Чтобы проинициализировать такой объект, ему надо присвоить что-то, типом которого будет файл. Для этого существует функция **open**(), которая возвращает объект файла. Она используется с двумя аргументами:

open(имя_файла, режим).

Первый аргумент - строка, содержащая имя файла, возможно с путем.

Функция корректно работает, если в путях используется удвоенный обратный слеш "\\" или одинарный прямой слеш "/".

Второй аргумент - символ, символизирующий процесс обмена

- 'r' - если файл будет открыт только для чтения,
- 'w' - если файл открыт только для записи,
- 'a' - если файл открыт для добавления.
- Если аргумент режим опущен - предполагается, что он равен '**r**'.

Стоит только описать объект как файл (присвоить ему значение при помощи функции **open**), как тут-же у него появляются методы, при помощи которых осуществляется файловый ввод - вывод.

В обычном случае файлы открываются в текстовом режиме - это значит, что вы читаете из файла и записываете в файл строки в определённой кодировке (по умолчанию используется CP1251 в Python 2.7.x и UTF-8 в Python 3.x

Итак, пусть f это объект типа файл.

f = **open**("c:\\t.txt",'w') - это мы открыли файл c:\t.txt для записи.

Для записи в файл существует следующий метод:

f.**write**(строка) записывает содержимое одной строки в файл. Несколько строк одним методом **write**, подобно **print**, вывести никак не получится. В Python 3.x этот метод возвращает количество записанных байтов. Заметим, что если мы вызовем метод **write** несколько раз, то в файле между записанными строчками перевода строк не будет. Если мы желаем делать переводы, то должны об этом позаботится сугубо, выводя в нужные места специальную строку "\n".

Не забываем закрывать файл методом f.**close**()

f = **open**("c:\\t.txt",'r') - это мы открыли файл c:\t.txt для чтения.

Для чтения содержимого файла существуют следующие методы:

f.**read**([N]) - метод читает некоторое количество данных и возвращает их в виде строки. N - необязательный числовой параметр, определяющий количество байт, которое будет прочитано за раз. Если этого параметра нет, то будет прочитано и возвращено всё содержимое файла, а если размер файла по величине окажется во много раз больше оперативной памяти вашего компьютера, то это уже будет ваша личная трагедия. Как только будет достигнут конец файла, f.**read**() вернёт пустую строку.

f.**readline**() - читает одну строку из файла и символ перевода строк "\n". Если f.**readline**() возвращает пустую строку, то значит, достигнут реальный конец файла, в то же время незаполненная строка, представленная посредством '\n', непуста.

f.**readlines**([N]) - возвращает список, содержащий все строки, находящиеся в файле. Если есть необязательный параметр N, то метод читает из файла указанное количество байт, плюс еще некоторое количество байт для завершения строки, и формирует список строк. Этот Метод используется для построчного чтения больших файлов, для того, чтобы не надо было загружать файл в память полностью.

Не забываем закрывать файл методом f.**close**()

Свои собственные функции

Функции в программировании это изолированный блок кода, обращение к которому в процессе выполнения программы может быть многократным.

Определение функции начинается с ключевого слова **def**, после которого следуют имя функции и, в скобках, аргументы. Если функция должна воспринимать несколько аргументов, то они перечисляются через запятую. Если у функции нет аргументов, то их нет. Но скобки быть обязаны.

Тип возвращаемого значения функции определяется при выполнении инструкции **return**, эта инструкция возвращает указанное в ней значение. Если инструкции **return** нет, то функция возвращает специальное значение - **None**.

Границы кода текста, относящегося к функции определяются операторными скобками, т.е. отступами. Оператор **return** обычно помещают в конце кода функции, хотя это не обязательно. Более того, оператор **return** вообще необязателен. Тогда функция становится аналогом того, что в других языках именуют подпрограммой.

Описание собственной функции:

def Имя_функции ([[форм_параметр_1], форм_параметр_2,] ...):

блок_кода_1

[**return** какое-нибудь_значение]

Вызов собственной функции:

x= Имя_функции ([[факт_параметр_1], факт_параметр_2,] ...)

Важные различия Python 2.7.x и Python 3.x

	Python 2.7.x	Python 3.x
Вывод на экран:	оператор **print** **print** "мама мыла раму"	функция **print** **print** ("мама мыла раму")
Ввод с клавиатуры	функция **input**() дает число, функция **raw_input**() строку	функция **input**() дает строку, ввода числа нет.
Деление двух целых чисел	Результат - целое число 3/4 = 0	Результат - действительное число 3/4 = 0,75
Русские буквы	кодировка CP1251 ("мама мыла раму" во всех смотрелках)	UTF 8 ("PjP°PjP° PjC‹P»P° CЂP°PjCѓ" в некоторых смотрелках)
Способ создания списка целых чисел	**range**() - итератор. Может использоваться как в операторе **for**, так и самостоятельно (создает весь список)	**range**() - генератор, используется только в операторе **for**. (генерирует только очередное значение)

Источники информации

http://pythontutor.ru
http://pythonworld.ru
https://docs.python.org/2.7
https://docs.python.org/3.5
https://ru.wikibooks.org/wiki/Python
http://itnote.ru/2011/05/25/python-file
http://www.iakovlev.org/index.html?p=5791
http://wombat.org.ua/AByteOfPython/toc.html
http://radio-hobby.org/modules/instruction/python
http://python-b.blogspot.ru/2012/09/blog-post_15.html
http://toly-blog.ru/programming/python-recepty-massivy
http://www.py-my.ru/post/4bfb3c691d41c846bc000050#!
http://santysoft.narod.ru/articles/art_momentary_python.html
http://www.russianlutheran.org/python/idle_doc/using_idle.html
http://younglinux.info/sites/default/files/python_structured_programming.pdf
http://bit126.moy.su/load/programm/python/uslovija_i_cikly_v_python/72-1-0-150

Содержание

Среда 2
Элементы языка (Операторы, функции, объекты) 3
Типы объектов. 4
Числовые переменные 5
Строки 6
Списки 8
Двумерные массивы 10
Логические выражения и логический тип данных 10
Об операторных скобках 11
Условный оператор 11
Оператор Цикла 11
Ввод-вывод 12
Файловый ввод-вывод 13
Свои собственные функции 14
Важные различия Python 2.7.x и Python 3.x 15