Python. Краткий справочник для учителя информатики.

О. М. Киселев
ok@ufanet.ru

28 апреля 2017 г.

Оглавление

Глава 1

Введение

1.1 Предисловие

Большинство языков программирования имеют общие типы данных и конструкции. На самом деле это диалекты протоязыка, на котором мыслят программисты.

Этот справочник сделан после чтения небольшого курса из четырех лекций экспертам ЕГЭ по введению в язык Python в Башкирском институте развития образования, зимой 2017 года.

Цель справочника – привести базовые конструкции языка. Помочь учителям, не обучающим языку Python, разбираться в коде учеников, которые на этом языке решают задачи по информатике.

- Справочник для людей, знакомых с не одним языком программирования.
- Язык Python унаследовал многие конструкции языков программирования, созданных ранее.
- Описывать структуру языка знакомым с программированием людям начиная с азов – бессмысленная трата времени. Почти все окажется знакомым или очень хорошо знакомым.
- Поэтому будет большое количество умолчаний.
- Основная цель – сфокусироваться именно на особенностях языка, считая, что основные структуры данных и операторы языка понятны.

1.2 Немного формальностей

- Python язык интерпретируемый, но есть для него и компиляторы.
- Существуют две развивающиеся ветки языка
  - "Старая"ветка Python 2.x с большим количеством библиотек
  - "Молодая"(с 2008 года) ветка Python 3.x
  - Программы на Python 3.x и на Python 2.x, вообще говоря, несовместимы. Можно считать Python 2.x и Python 3.x разными языками.

Список литературы по языку Python очень обширен. При подготовке лекций были использованы несколько книг основополагающего характера:

1 Learning Python, by Mark Lutz [2];

2 Rapid GUI Programming with Python and Qt. The Definitive Guide to PyQt Programming by Mark Summerfield [3]

3 Thinking in Python Design Patterns and Problem-Solving Techniques, by Bruce Eckel [1].

1.3 Стиль программирования

- Язык Python поддерживает ООП. Все в нем является объектом.
- На языке Python легко писать программы в стиле функционального программирования.
- Синтаксис языка прост и понятен.
- Короткие программы легко читаемы.

1.4 Интерактивный режим

1.4.1 Интерпретатор

Интерпретатор Python работает в интерактивном режиме. Для знакомства с возможностями удобно воспользоваться функцией *помощи*

```
>>>help()
```

```
четыре абзаца текста
help>
```

В ответ будет предложено ввести ключевое слово или название модуля.

Если помощь не нужна, можно нажать ENTER, чтоб вернуться в режим интерпретатора. Если нужна – ввести ключевое слово, по которому требуется помощь.

Выйти из режима помощи в режим ожидания ввода ключевого слова можно, нажав клавишу 'q'. Далее – в режим интерпретатора – нажав ENTER.

Глава 2

Базовые типы данных

2.1 Комментарии и пробелы

Первое с чего нужно начать – с того, что никак не повлияет на Вашу программу:

- #*-знак комментария* длиной до конца текущей строки.

```
>>> 123*456  # интерпретатор можно использовать
как калькулятор
56088
```

- ; можно использовать в конце строки, но не обязательно.

```
>>> 2*2;
4
>>> 2*2
4
```

- Строки начинаются с первой позиции слева.

- *Отступы* в левой части строки определяют блоки операторов внутри программы. Если строка начинается с неожиданного для интерпретатора отступа, появится сообщение об ошибке:

```
>>>  x=2 #строка начинается с  пробела
  File "<stdin>", line 1
    x=2
    ^
IndentationError: unexpected indent
```

- Пробелы внутри строки игнорируются.

```
>>>2 *  2 #пробел между * и цифрой 2
4
```

- Строки заканчиваются символом перевода каретки (клавиша Enter).

2.2 Идентификаторы

- *Идентификатором* в Python 2.x может быть любая последовательность ASCII символов, если она не начинается с цифры и не содержит пробелов и символов арифметических операций.

```
>>> a123=5 #корректное имя идентификатора
>>> _23=4 #корректное имя идентификатора
>>> 3w='www' #имя начинается с цифры
  File ''<stdin>'', line 1
    3w='www'
    ^
SyntaxError: invalid syntax
```

- В Python 3.x в качестве идентификаторов можно использовать символы UTF8.

```
>>> имя_переменной ='кириллица'
```

2.3 Объект None

В Python введен объект **None**. По своей сути это указание на ничто. Может использоваться при инициализации переменных, при обработке исключений и других похожих случаях.

```
>>> y=None
>>> y
>>> id(y)
3077446216
>>> None==False
False
```

2.4 Числовые типы

В Python к числовым типам данных относятся:

- *булевы (bool)*,
- *целые (int)*,

- *числа с плавающей точкой (float),*
- комплексные (complex).

2.4.1 Булевы числа

принимают значения **True** и **False**, но могут использоваться и как числа 0 и 1. Для того чтобы объявить булеву переменную нужно связать значение имени переменной с булевым объектом:

```
>>> булеваПеременная=True
>>> repr(булеваПеременная)
'True'
>>> булеваПеременная-1
0
>>> булеваПеременная==1
True
```

2.4.2 Целые числа

- В Python 2.x
  - int реализованы с помощью типа long в Си от -2147483647 до 2143483647

  - long int – целые неограниченной длины

- В Python 3.x

  - все целые числа могут иметь длину, ограниченную только памятью, доступной для интерпретатора

2.4.3 Литералы для целых чисел

- Десятичная запись – начинается с любой цифры кроме нуля.

- Двоичная запись – начинается с **0b**, дальше нули и единицы.

```
>>> bin(15)
'0b1111'
```

- Восьмеричная – начинается с **0o**, дальше любые цифры от 0 до 7

```
>>> oct(15)
'0o17'
```

- Шестнадцатеричная – начинается с **0x** или **0X**, дальше 0-9 и A-F или a-f

```
>>> hex(15)
'0xf'
```

- Если запись выходит за пределы допустимого диапазона – генерируется исключение **OverflowError**

2.4.4 Литералы для чисел в Python 2.x

- в Python 2.x восьмеричная запись числа может начинаться с **0**,

- в Python 2.x длинное целое заканчивается символом **L** или **l**, рекомендуется **L**, чтоб не путать с единицей.

2.5 Числа с плавающей точкой и комплексные числа

2.5.1 Числа с плавающей точкой

Числа с плавающей точкой реализованы на основе Си double.

- Не менее 10 значащих цифр.
- Максимальное значение 10**308

```
>>> 1E308
1e+308
>>> 2E308
inf
```

- Минимальное значение по абсолютной величине

```
>>> 2**(-1024-50)
5e-324
>>> 2**(-1024-51)
0.0
```

2.5.2 Комплексные числа

- Комплексное число можно записать в виде суммы его вещественной и мнимой частей.

```
>>> 2+1j
(2+1j)
```

- Мнимая часть комплексного числа должна заканчиваться символом 'j':

```
>>> 1j
1j
>>> 1j*1j
(-1+0j)
```

- Комплексное число Z имеет атрибуты Z.**real** и Z.**imag**

```
>>> (123+456j).real
123.0
>>> (123+456j).imag
456.0
```

2.6 Арифметические операции

- $a**b$ – возведение в степень

```
>>> 3.14**2.71
22.21668954600232
```

- a% b – остаток от деления

```
>>> 3.14 % 3
0.14000000000000012
```

- a//b – деление нацело

```
>>> 3.14//2
1.0
```

2.6.1 Операторы присвоения и преобразования

- Оператор присвоения не возвращает значение

```
>>>x=1
>>>x
>>>1
```

- Оператор множественного присвоения производит присвоение "параллельно":

```
>>> x=1
>>> a=2
>>> x,a=a,x
>>> x
2
>>> a
1
```

- операторы преобразования +=, -=, *=, /=, и %=.

```
>>> a=5
>>> a+=2
>>>a
7
>>> a%=3
>>> a
1
```

2.6.2 Побитовые операции над целыми числами

- a|b – побитовое или

```
>>> bin(0b111|0b1001)
'0b1111'
```

- a^ b – исключающее или

```
>>> bin(0b111^0b1001)
'0b1110'
```

- a&b – побитовое умножение

```
>>> bin(0b111 & 0b1001)
'0b1'
```

- $a << n$ – сдвиг влево на n позиций

```
>>> bin(0b111 << 3)
'0b111000'
```

- $\sim$ символ побитового отрицания. Меняет все биты в представлении числа, включая бит, определяющий знак.

```
>>> ~ 0b0
-1
>>> bin(-1)
'-0b1'
```

2.7 Строки

- Строки заключаются в одинарные или двойные кавычки.
- В Python 3.x все строки состоят из восьмибитных символов.

```
>>> 'обычная строка в Python3'
'обычная строка в Python3'
```

- В Python 2.x обычные строки состоят из семибитных символов.

```
>>> 'A 7-bit string'
'A 7-bit string'
```

- В Python 2.x строки из восьмибитных символов начинаются с символа **u**

```
>>> u'8 бит'
u'8 \backslash u0431 \backslash u0438
\backslash u0442'
```

2.7.1 Операции со строками

- Доступ по номеру символа в строке

```
>>> "строка"[0]
'с'
```

- Для доступа к символу с конца строки используйте отрицательные числа

```
>>> "строка"[-3]
'о'
```

- Умножение строки на целое число

```
>>> "строка"*3
'строкастрокастрока'
>>> "строка"*(-3)
''
```

- Слияние строк

```
>>> "сумма"+"строк"
'суммастрок'
```

- Длина строки

```
>>> len("длина строки")
12
```

- Строка – **неизменяемый** объект, то есть нельзя в строке заменить символ:

```
>>> S="корона"
>>> S[4]="в"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support
item assignment
```

2.7.2 Срезы

- Части строк можно выделять срезами:

```
>>> "строка"[1:4]
'тро'
>>> "строка"[:4]
'стро'
>>> "строка"[3:]
'ока'
```

- С помощью среза можно создать новую строку, заменив символы в старой

```
>>> S="корона"
>>> S=S[:4]+"в"+S[-1]
>>> S
'корова'
```

2.8 Список

Список (**List**) – последовательность данных разных типов:

```
>>> L=[1,'элемент',['вложенный','список']]
>>> L[2]
['вложенный', 'список']
>>> L[1]
```

```
'элемент'
```

Список – изменяемый тип данных. Его можно сортировать, можно добавлять и удалять элементы:

```
>>> L.append('(x+1)/x=1+1/x')
>>> L
[1, 'элемент', ['вложенный', 'список'],
'(x+1)/x=1+1/x']
>>> L.pop(2)
['вложенный', 'список']
>>> L
[1, 'элемент', '(x+1)/x=1+1/x']
```

2.8.1 Сортировка

Для списка есть встроенный метод сортировки

```
>>> L=[2,3,1,4,7,5,9,0]
>>> L.sort()
>>> L
[0, 1, 2, 3, 4, 5, 7, 9]
```

Список с разнородными типами данных сортировке не поддаётся:

```
>>> L=['No','First','последний','Да']
>>> L.sort()
>>> L
['First', 'No', 'Да', 'последний']
>>> L.append(1)
>>> L
['First', 'No', 'Да', 'последний', 1]
>>> L.sort()
Traceback (most recent call last):
  File ''<stdin>'' , line 1, in <module>
TypeError: unorderable types: int() < str()
```

2.9 Кортеж

Кортеж (**tuple**)– неизменяемый список

```
>>> T=('алгол','фортран','лисп','пл/1')
>>> T.index('лисп')
2
>>> T.count('перл')
0
>>> T+('перл','кобол')
('алгол', 'фортран', 'лисп', 'пл/1', 'перл',
'кобол')
```

При формировании кортежа из двух и более элементов скобки можно не использовать

```
>>> кортеж='БЭСМ-6','НАИРИ'
>>> кортеж
('БЭСМ-6', 'НАИРИ')
```

2.10 Словарь

Словарь (**dict**) – это набор пар ключ:значение

```
>>> FORTRAN={'название':'FORmul TRANSlation',
...'дата создания':1954,
'применение':'научные и инженерные расчеты'}
>>> FORTRAN['применение']
'научные и инженерные расчеты'
>>> FORTRAN['применение']+=', статистика'
>>> FORTRAN['применение']
'научные и инженерные расчеты, статистика'
```

Добавим пару ключ:значение (список).

```
>>> FORTRAN['версии']=['66','77','90','95',
'2003','2008']
```

Ключ должен быть уникальным. Ключом в словаре может быть только неизменяемый тип:

```
>>> Квадраты={1:1,2:4}
>>> z=3
>>> Квадраты[z]=9 #добавлена пара 3:9
>>> Квадраты
{1: 1, 2: 4, 3: 9}
>>> y=[4,5,6] #список чисел
>>> Квадраты[y]=[16,25,36] # добавим список (ошибка!)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

Используя список ключей и функцию **sort()**, можно сортировать словари.

2.11 Множество

Множество (**set**) – набор объектов различных типов. Элементы множества должны быть уникальны.

```
>>> типы_данных_Python3={'Boolean','int','float',
```

```
'complex','string','list','tuple','dictionary',
'set'}
>>> типы_данных_Java={'byte','short','int','long',
'float','character','Boolean','string','array'}
```

2.11.1 Операции над множествами

- Пересечение

```
>>> типы_данных_Python3 & типы_данных_Java
{'int', 'float', 'string', 'Boolean'}
```

- Объединение

```
>>> типы_данных_Python3 | типы_данных_Java
{'set', 'string', 'dictionary', 'tuple', 'int',
'character', 'float', 'list', 'long', 'short',
'complex', 'array', 'byte', 'Boolean'}
```

- Разность

```
>>> типы_данных_Python3 - типы_данных_Java
{'set', 'list', 'complex', 'dictionary',
'tuple'}
```

- Подмножество

```
>>> типы_данных_Python3 > типы_данных_Java
False
```

2.12 Расширения типов данных

Существуют модули, расширяющие набор данных. Например:

- модуль **decimal**, который вводит числа с фиксированным числом знаков после десятичной запятой,
- модуль **fractions**, позволяет делать вычисления в поле рациональных чисел.

Глава 3

Связывание и присвоение

3.1 Неизменяемые типы данных

В Python есть разделение на изменяемые (**mutable**) и неизменяемые (**immutable**) типы данных.

К неизменяемым относятся **числа, строки, кортежи**. Неизменяемые объекты можно только заменить другим объектом.

```
>>> S='строка- неизменяемый тип данных'
```

```
>>> S[4]='K'
Traceback (most recent call last):
  File ''<stdin>'', line 1, in <module>
TypeError: 'str' object does not support
item assignment
>>> T=('элементы','кортежа','нельзя', 'поменять')
>>> T[2]='можно'
Traceback (most recent call last):
  File ''<stdin>'', line 1, in <module>
TypeError: 'tuple' object does not support
item assignment
```

3.2 Изменяемые типы данных

К изменяемым относятся **списки, словари, множества**. В изменяемых типах можно менять содержимое. Например, можно поменять элемент множества или элемент списка.

```
>>> L=['элементы','списка','нельзя', 'поменять']
>>> L[2]='можно'
>>> L
['элементы', 'списка', 'можно', 'поменять']
>>> M={'множество','можно', 'расширить'}
```

```
>>> M.add('новым элементом')
>>> M
{'новым элементом', 'можно', 'множество',
'расширить'}
```

3.3 Динамическая типизация и присваивание

3.3.1 Динамическая типизация.

Одной и той же переменной могут быть последовательно присвоены значения разных типов. Это не приводит к ошибке.

```
>>> x='string'
>>> x={'s','e','t'}
```

После связывания имени переменной с объектом {'s','e','t'}, объект 'string' удаляется из памяти уборщиком мусора, так как он не связан ни с одной переменной.

В то же время Python – язык со строгой типизацией:

```
>>> y=('t','u','p','l','e')
>>> x+y #попытка сложиь объекты разных типов
Traceback (most recent call last):
  File ''<stdin>'', line 1, in <module>
TypeError: unsupported operand type(s) for +:
'set' and 'tuple'
```

3.3.2 Присваивание

. При создании объекта изменяемого типа создается ссылка, а не новый объект:

```
>>> L=['л','о','ж','к','а']
>>> копия_L=L #новый объект? Проверим!
>>> копия_L
['л', 'о', 'ж', 'к', 'а']
>>> L[2]='ш'
>>> L[0]='k'
>>> копия_L
['k', 'о', 'ш', 'к', 'а'] #Нет! Ссылка на объект  L
>>> id(L)==id(копия_L)
True
>>> L=['н','о','ж','к','а'] #Вот теперь создаем
новый объект!
```

```
>>> копия_L # сслыка на старый остается!
['л', 'о', 'ж', 'к', 'а']
```

Создадим список, совпадающий по содержанию с другим списком, но не являющийся ссылкой:

```
>>> L
['н', 'о', 'ж', 'к', 'а']
>>> новый_список=list(L)
>>> новый_список
['н', 'о', 'ж', 'к', 'а']
>>> L[0]='л'
>>> L # список L изменился
['л', 'о', 'ж', 'к', 'а']
>>> новый_список # не изменился
['н', 'о', 'ж', 'к', 'а']
```

3.3.3 Присваивание для неизменяемых типов

При замене объекта неизменяемого типа создается новый объект, а не ссылка:

```
>>> K=('л','о','ж','к','а') # строка-- неизменяемый
тип
```

```
>>> копия_К=К
>>> id(К)==id(копия_К)
True
>>> К=('к','о','ш','к','а')
>>> id(К)==id(копия_К)
False
>>> копия_К
('л', 'о', 'ж', 'к', 'а')
```

3.3.4 Изменяемые данные внутри неизменяемого типа

Кортеж – неизменяемый тип данных, но элементом кортежа может быть, например, список – изменяемый тип.

```
>>> К=(['л','о','ж'],'к','а')#первый элемент
-- список
>>> К
(['л', 'о', 'ж'], 'к', 'а')
>>> К[0]=['м','о','ш'] # заменить первый
элемент нельзя!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support
item assignment
>>> K[0][0]='м' #изменение первого элемента
списка
>>> K[0][2]='ш'#изменение третьего элемента
списка
>>> K #результат-- кортеж не изменился,
но изменился список!
(['м', 'о', 'ш'], 'к', 'а')
>>>
```

Заменить элемент кортежа нельзя, но изменить элемент кортежа, вообще говоря, можно!

3.3.5 Множественное присваивание

```
>>> [a,b]=1,2
>>> a
1
>>> b
2
>>> a,b=1,2,3 #справа слишком много значений
Traceback (most recent call last):
  File ''<stdin>'', line 1, in <module>
```

```
ValueError: too many values to unpack (expected 2)
>>> a,b=1,2
>>> a,b
(1, 2)
```

Глава 4

Операторы языка

4.1 Ключевые слова

Ключевые слова Ключевые слова Python:

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

4.2 Условный оператор

Условное ветвление – оператор **if**.

В операторе используется выражение сравнения с помощью символов сравнения: <, >, <=, >= ,==, !=. Затем следует блок операторов, который выполняется, если выражение сравнения верно. Блок операторов определяется одинаковым числом пробелов в начале каждой строки этого блока.

```
if выражение сравнения: #ниже -- блок кода
    операторы # пробелы в начале -- блок кода
elif выражение сравнения:
    операторы # стиль -- четыре пробела
else:
    операторы
```

Если оператор одиночный – можно в одной строке

```
if выражение сравнения: оператор
```

4.3 Логические операторы

Логические операторы

Цепное сравнение

```
>>> -3<-2<-1 #цепное сравнение
True
>>> (-3<-2)<-1 # вычисление выражения и
последующее сравнение
False
>>> 1==2<3 #цепное сравнение
False
>>> (1==2)<3 # вычисление выражения и
последующее сравнение
True
```

Цепное сравнение можно использовать в операторах ветвления:

if 1<x and x<10: # эквивалентно

if 1<x<10:

Логические операторы **and, or** можно заменять на побитовые операции **&** (and), | (or).

```
>>> True & True
True
>>> True & False
False
>>> False & False
```

```
False
>>> True | True
True
>>> True | False
True
>>> False | True
True
```

Побитовое отрицание $\sim$ — не эквивалентно отрицанию **not**:

```
>>> not True==False
True
>>> ~- True==False #
True
>>> not False==True
True
>>> ~- False==True
False
```

Причина неэквивалентности кроется в представлении **False** и **True** как целых чисел:

```
>>> False==0
True
```

```
>>> True==1
True
```

и компьютерном представлении отрицательных целых чисел в двоичной системе.

```
>>> ~-1
0
>>> ~-True
0
>>> ~-0
-1
>>> ~-False
-1
```

4.3.1 Идентичность, включение

Идентичность **is, is not** удобна для определения ссылок на один объект

```
>>> a=1
>>> b=c=a
>>> b is c
True
```

Включение **in, not in** удобно для списков, словарей и строк.

```
>>> 'версии' in FORTRAN
True
>>> FORTRAN['версии']
['66', '77', '90', '95', '2003', '2008']
```

4.4 Цикл while

Циклы **while**

```
while выражение:
    тело цикла # начинается с пробелов
```

все строки в теле цикла начинаются с одинакового количества пробелов.

Цикл можно прервать оператором **break** и выйти из цикла или перейти к следующему проходу цикла, воспользовавшись **continue** пропустив оставшиеся операторы тела цикла, в примере это **тело цикла (часть 3)**:

```
while выражение:
   тело цикла (часть 1)
```

```
   if условие: break #если необходимо
   тело цикла (часть 2)
   if условие: continue #если необходимо
   тело цикла (часть 3)
```

Цикл можно связать с оператором **else**

```
while  выражение:
   тело цикла
   if условие: break
   тело цикла
else:
   блок операторов
```

В этом случае блок операторов после **else** выполняется при нормальном завершении цикла и не выполняется при завершении цикла по условию **break**.

```
>>> i=-1
>>> while i<10:
...     if i==-1: break
...     i+=1
... else:
...   i=-10
...
```

```
>>> i
-1
```

Здесь **break** позволяет и выйти из цикла и обойти операторы внутри **else**.

4.5 Цикл for

```
for переменная in итерация:
   тело цикла
   if условие: continue #если необходимо
   тело цикла (часть 3)
   if условие: break #если необходимо
   тело цикла
else: #не обязательно
   блок операторов
```

```
>>> for x in 'два':
...     print(x)
...
д
в
а
```

```
>>> for x in range(10):
...   print(x)
```

Здесь **range**(a[,b[,c]]) – функция, выдает последовательность целых чисел от **a** до **b** с шагом **c**

4.5.1 Генератор списка

Цикл **for** удобно использовать как генератор списка:

```
>>> кратные_5=[x for x  in range(0,50,5)]
>>> кратные_5
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

4.6 Исключения

Блок кода под **try** выполняется, если не возникнет исключения.

Блок кода под **except** выполняется, если возникло исключение.

Блок кода под **finally** выполняется в любом случае.

```
try:
    код_программы
except исключение:
    код_если_возникло_исключение
else:
    код_программы,_если_исключения_не_случилось
finally:#если необходимо
    код программы #выплняется всегда
```

Пример:

```
>>> try:
...     10/0
... except ArithmeticError:
...     print('Деление на нуль')
...
Деление на нуль
```

Список исключений **ArithmeticError, IOError, OSError, EOFError, IndexError, KeyError, ValueError**

Глава 5

Функции

5.1 Объявление функций

Интерпретатор Python выполняет инструкции последовательно. Поэтому функции рекомендуется определять в начале файла, содержащего программу

```
def имя_функции(необязательный_список_параметров):
    тело функции
    return результат #необязательный оператор
```

Функция всегда возвращает значение, например **None**, если оператор **return** не используется.

Функции с различным числом аргументов можно объявлять так:

```
def имя_функции(arg0=None,arg1=None,arg2=None):
    if arg0 is not None:
        тело условного оператора
    if arg1 is not None:
        тело условного оператора
```

Замечание: так как **None** – глобальный объект, то сравнение с ним проводится быстро.

5.2 Неименованные функции

Python позволяет создавать неименованные функции – лямбда функции

```
имя_функции= lambda аргумент: выражение_в_строку
```

Ключевое слово **lambda** позволяет создавать неименованную функцию. Такая функция не может содержать управляющих структур – ветвлений и циклов и не содержит **return**. Значение – результат вычисления оператора в теле функции.

```
>>> SinT= lambda x: x-x**3/6+x**5/(120)
>>> SinT(3.14/6)
0.49977222768366597
```

5.3 Модули

- Модуль – любой файл на Python.
- Модули обычно имеют расширение .py
- Модуль загружается командой:

  ```
  import имя_модуля #загружается весь модуль
  ```

  Обращение к переменным и функциям модуля :

  ```
  имя_модуля.имя_функции(агрумент)
  имя_модуля.имя_переменной
  ```

```
>>> import math
>>> math.sin(3.14/2)
0.9999996829318346
>>> sin(3.14/2)
```

```
Traceback (most recent call last):
  File ''<stdin>'', line 1, in <module>
NameError: name 'sin' is not defined
```

- Функция или переменная из модуля загружается командой:

  ```
  from модуль import функция_или_переменная
  ```

  При таком способе загрузки к загруженной переменной или функции можно обращаться напрямую. Но имя загруженной переменной или функции может заместить уже существующее в Вашей программе.

  ```
  >>> from math import sin
  >>> sin(3.14)
  0.0015926529164868282
  ```

- Модуль можно загрузить только раз. Второй раз вызов **import** ничего не даст

- Для перезагрузки модуля можно воспользоваться функцией **reload()** из модуля **imp** :

```
from imp import reload #импортируется
функция reload()
reload(sys) # перезагружается модуль sys
```

5.4 Ввод данных с клавиатуры

Для ввода данных используется функция **input()**. Она возвращает все символы введенные с клавиатуры до нажатия клавиши **ENTER**:

```
>>> x=input()
2
>>> x
'2'
```

Для преобразования введенной строки, например, в числовой тип можно воспользоваться функциями приведения типов:

```
>>> x=int(input())
2
>>> x
2
```

Глава 6

Классы

6.1 Конструкции объектно-ориентированного программирования

Язык Pyhon позволяет использовать ООП.

- Классы в Python позволяют описывать атрибуты и функции, связанные с объектом.
- Скрывать данные и методы.

- Наследовать атрибуты и методы родительских классов.
- Переопределять атрибуты и методы родительских классов.

6.1.1 Видимость имен

- Пространство имен внутри класса – локальное.
- Для обращения к атрибутам экземпляра класса.

```
имя_экземпляра_класса.название_атрибута
```

- Для обращения к методам экземпляра класса.

```
имя_экземпляра.метод(список_аргументов)
```

- Внутри методов класса атрибуты этого же класса доступны по ссылке

```
self.имя_атрибута
```

6.2 Создание класса

Любой создаваемый класс обязательно имеет класс-родитель. Хотя бы базовый класс object.

Создание класса в два этапа:

- объект конструируется

  ```
  __new__() #используется редко
  ```

- потом инициализируется

  ```
  __init__()
  ```

  используется для инициализации атрибутов, если это необходимо;

- атрибуты класса могут быть инициализированы методом

  ```
  __init__(self,атрибут1=значение,
  атрибут2=значение)
  ```

6.2.1 Пример шаблона для создания класса

```
class имя(базовый_класс_1,базовый_класс_2):
    атрибут1=значение
    атрибут2=значение
    def  __init__}(self,name)
        self.name=name
    def  __скрытый_метод_(self,
    список_аргументов)
        тело метода
    def метод1(self, список_аргументов):
        тело метода
    def метод2(self, список_аргументов):
        тело метода
```

Методы с двумя подчерками в начале и хотя бы одним подчерком в конце скрыты от внешнего доступа. Для них есть специальные методы доступа.

В Python есть соглашение об аналоге приватных методов и атрибутов – методы и атрибуты, начинающиеся с одного символа подчеркивания. Они не являются приватными, но считается, что могут быть изменены без всякого уведомления для тех

кто использует класс, например как класс из некоторой библиотеки.

Имя **self** используется для того, чтобы при оперировании внутри класса в методах можно было использовать атрибуты (переменные), используемые именно в этом классе.

Вместо **self** можно использовать любой другой идентификатор.

6.3 Пример. Базовый класс.

```
class уравнение( object):
    к=None
    корень=None
    def __init__(self,коэффициенты):
        self.к=коэффициенты
    def решение(self):
        return None
```

6.3.1 Пример. Производный класс 1.

```
class линейное_уравнение(уравнение):
    def решение(self):
```

```
        try:
            return self.к[1]/self.к[0]
        except ArithmeticError:
            return  'нет решения'
```

6.3.2 Пример. Производный класс 2.

```
class квадратное_уравнение(уравнение):
  def решение(self):
    try:
      if ( self.к[0]+ self.к[1]+ self.к[2]==0):
        return  [1, self.к[2]/ self.к[0]]
          elif ( self.к[0]- self.к[1]+
          self.к[2]==0):
            return [-1,- self.к[2]/ self.к[0]]
              else:
                return 'решение неизвестно'
     except  ArithmeticError:
       return 'Ошибка в вычислениях'
     else:
       return 'неизвестная ошибка'
```

6.3.3 Пример. Реализация.

```
ур1=линейное_уравнение([0,1])
print (ур1.решение())
ур2=квадратное_уравнение([1,-2,1])
print (ур2.решение())
```

6.4 Доступ к атрибутам класса

Для доступа к атрибутам класса и для изменения атрибутов можно использовать методы

```
def установить_имя_атрибута( self, значение):
    self.имя_атрибута=значение

def считать_имя_атрибута( self, значение):
    self.имя_атрибута
```

В Python есть специальный способ для прямого доступа к атрибутам класса с помощью метода **property()**.

Существуют и другие предопределенные методы для классов и их атрибутов.

Литература

[1] Bruce Eckel. *Thinking in Python Design Patterns and Problem- Solving Techniques.* MindView, Inc., 2002.

[2] Mark Lutz. *Learning Python, Fifth Edition.* O'Reilly, 2013.

[3] Mark Summerfield. *Rapid GUI Programming with Python and Qt. The Definitive Guide to PyQt Programming.* Prentice Hall, 2009.

Предметный указатель