Технологии програм-ия. Python как второй язык.

Лекция 2/13

Общая информация: *help-online*, похожесть на C в ключевых словах и символах.

Модель данных, объекты: иерархия типов, некоторые основополагающие понятия, разбор типов.

Функционал: встр-ые функции, управляющие инструкции.

Типы коллекций (контейнерные): список – самый фундаментальный тип.

Полезное: полезные модули. Вопросы для экз. (sic!).

(v. 0.6)

И.Ф.Травов

igor.travov@gmail.com

Саров 2017/09 - 2018/05

Оболочки Питона: python3[†], ipython3 (выбирай), процесс сдачи лаб – в оболочке Питона (пусть в Pycharm)...

```
python    # запуск интерактивной оболочки (# - комментарий)
Python 3.6.2 (default, Aug 11 2017, 11:59:59)
[GCC 7.1.1 20170622 (Red Hat 7.1.1-3)] on linux
Type "help", "copyright", "credits" or "license" for more infor
# три символа >>> есть приглашение в оболочке
#  ... - приглашение для внутрикодового блока, ентер - конец
>>> help()    #обязательно со скобками, запуск встроенного help
```

[†]если в Вашем дистрибутиве по умолчанию запускается питон2, необходимо прописать в .bashrc установочном файле, хоть в самом конце, строчку alias python='python3.6' и перезапустить этот файл, например вот так:
. .bashrc
Тогда при запуске python, будет всегда питон3.

встроенный help() в python3, там объяснения по ключевым словам, символам, модулям, топикам

```
>>> help()
Welcome to Python 3.6's help utility!

If this is your first time using Python, you should definitely
check out the tutorial at http://docs.python.org/3.6/tutorial/.

To get a list of available modules, keywords, symbols, or
topics, type "modules", "keywords", "symbols", or "topics".
Each module also comes with a one-line summary of  what it does
...
help> # вход в хелп окружение, меняется приглашение
help> keywords  # посмотрим кивёрды, символы, топики
```

help> keywords (всего их 33, похожи на Си, но 'switch'-а нет), в др языках по-разному: в C - 32, в Go - 25, в Ruby - 41, в Java - 49, в JS - 60, в C++ - 85... Гвидо не сторонник их менять, но, похоже, добавит в Python3.7 async и await для описания сопрограмм(см Pyth3.7.0a2)

```
help> keywords
```

False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

```
help> symbols
```

'что-то=' дополненное (augmented) присваивание, арифм-е, сравнит-е операции (в т.ч.побитовые) выполняются соотв-щими операторами. '...' – объект-одиночка ellipsis , '//' – деление нацело, '++' нет, '_' и '__' – в именах "спец" атрибутов, '()', '[]', '{}' – раздел-ли и операторы, ':' – объект слайс, '@' – декоратор, умножение матриц.

```
!=      *=      <<      ^
"       +       <<=     ^=
"""     +=      <=      _
%       ,       <>      --
%=      -       ==      `
&       -=      >       b"
&=      .       >=      b'
'       ...     >>      j
'''     /       >>=     r"
(       //      @       r'
)       //=     J       |
*       /=      [       |=
**      :       \       ~
**=     <       ]
```

help> topics – строго объясняют, полезны как справки

```
ASSERTION           DELETION            LOOPING          SHIFTING
ASSIGNMENT          DICTIONARIES        MAPPINGMETHODS   SLICINGS
ATTRIBUTEMETHODS    DICTIONARYLITERALS  MAPPINGS         SPECIALATTRIBUTES
ATTRIBUTES          DYNAMICFEATURES     METHODS          SPECIALIDENTIFIERS
AUGMENTEDASSIGNMENT ELLIPSIS            MODULES          SPECIALMETHODS
BASICMETHODS        EXCEPTIONS          NAMESPACES       STRINGMETHODS
BINARY              EXECUTION           NONE             STRINGS
BITWISE             EXPRESSIONS         NUMBERMETHODS    SUBSCRIPTS
BOOLEAN             FLOAT               NUMBERS          TRACEBACKS
CALLABLEMETHODS     FORMATTING          OBJECTS          TRUTHVALUE
CALLS               FRAMEOBJECTS        OPERATORS        TUPLELITERALS
CLASSES             FRAMES              PACKAGES         TUPLES
CODEOBJECTS         FUNCTIONS           POWER            TYPEOBJECTS
COMPARISON          IDENTIFIERS         PRECEDENCE       TYPES
COMPLEX             IMPORTING           PRIVATENAMES     UNARY
CONDITIONAL         INTEGER             RETURNING        UNICODE
CONTEXTMANAGERS     LISTLITERALS        SCOPING
CONVERSIONS         LISTS               SEQUENCEMETHODS
DEBUGGING           LITERALS            SEQUENCES
```

PyPI - the Python Package Index – ПайПи Индекс пакетов – сторонние библиотеки, архивируются по определенным правилам, всего более 110 тыс. и постоянно растут (1-2тыс_пакетов/месяц), https://pypi.python.org/pypi

PyPI: - the Python Package Index
is a repository of software for the Python programming language.
There are currently 112453 packages here.

Модель данных в языке

Встроенные типы (всего стандартных 13):

- если по классике тип – это набор значений данных и операций с ними, то в объектных языках сложнее:
- данные 'поднимаются' (абстрагируются) до уровня объектов,
- объекты имеют определённое состояние (в т.ч. в отношениях с другими) и поведение (возможно, многофункциональное) и т.о. "классического" типа для их характеристики уже мало;
- более того, объекты в Питоне это единственные строительные блоки и диктуют всё, и тип (type) у них не единственная характеристика, и правомочно говорить не о модели данных, с их "классическими" типами, а о модели объектов в Питоне.

Смотрим на эту модель объектов (иерархию встроенных типов) и пытаемся сравнить с Си-шными базовыми типами и их квалификаторами:

1. None (синглтон, одиночка, одно значение)
2. NotImplemented (синглтон)
3. Ellipsis (многоточие, синглтон)
4. numbers.Number (числовой)(int, bool, float, complex – неизмен-ые)
5. Sequences (последовательности) (упоряд-енная коллекция эл-тов с неотриц-ми индексами, с длиной и срезами. Неизменяемые: strings, tuples, bytes, ranges. Изменяемые: lists, byte arrays.)
6. Set types (множества) (неупоряд коллекция уник-ых безиндексных объектов, изм/неизменяемых, Sets/Frozen sets)
7. Mappings (отобр-я) (индекс-ся ключами по произвол множ эл-тов)
8. Callable types (через call оператор(): юзерн функции, методы экзем-ров классов, генераторные, корутинные, асинхр-генер-ые, встр-ые функции, встр методы, классы, экзем-ры классов)
9. Modules (основной организац-й блок Питоновского кода)
10. Custom classes (собств-ые классы, namespace класса в dict)
11. Class instances (экзем-ы кл-в, в т.ч. с нетрив повед-ем аттриб-в)
12. I/O objects (Input/Output, иначе файловые объекты)
13. Internal types (код-объекты, фрейм-об-ты, трассир-назад об-ты (исключ-ия), слайс-об-ты, статик-метода, класс-метода объекты)

```
>>> type(3)                    #см. п.4 на предыдущем  слайде
<class 'int'>                  #здесь объект -- число целое
>>> type(3.)
<class 'float'>                #здесь объект -- число плавающее
>>> def foo():                 #объявили функцию, которая
...   pass                     # ничего не делает
...
>>> type(foo)                  #см. п.8 на предыдущем слайде
<class 'function'>             #здесь объект -- функция
>>> fp = open('.bashrc')       #объявили файл и он в наличии
>>> type(fp)                   # (см. п.12 на пред слайде)
<class '_io.TextIOWrapper'>  # объект ввода/вывода
>>> class Aa:                 #объявили новый класс(для нового объекта)
...   pass
...
>>> type(Aa)                   #см. п.10 на предыдущем слайде
<class 'type'>  #здесь новый объект(новый тип через новыйкласс)
>>>
```

Объекты: ('все есть объект' – мантра Питона).

- ...все данные в Питоновской программе представлены объектами или отношениями между объектами. Код в памяти тоже представлен объектом...
- ...Каждый объект имеет 1)идентичность, 2)тип(или класс) и 3)значение.
- Первые два не меняются в течение своей жизни.
- Тип(см.2) объекта задаёт:
- действия, поддерживаемые объектом, т.е. его функциональность через методы,
- его (объекта) возможные значения(см.3) (через его атрибуты),
- значение у некот-х объектов изменяемое (mutable), у некоторых неизменяемое (immutable)...

Где видно иерархию объектов Питона, что под капотом?

Если подгрузить модуль builtins, и вызвать его help(builtins), то станет видно, что главным объектом из которого всё наследуется есть object.

```
  object               # под object идут сначала исключени
    BaseException      # (их где-то 65 вместе с варнингами)
      Exception        # далее в алфавитном порядке
....                   # от bytearray до zip 24 встр-ых
    bytearray          #  объекта
    bytes             # (24 класса наследников object),
    classmethod       # не считая Ellipsis, False, None,
    complex           # NotImplemented и True
    dict
...
    int
      bool
...
    tuple
    zip              # места не хватило для всех 24-х
```

Иерархия объектов в Питоне напоминает иерархию процессов UNIX, рождающихся от процесса init при запуске системы. (Понятно, с чем Гвидо работал в 80-е, то и заложил в язык.)

```
>>> help(object)
Help on class object in module builtins:
class object
 | The most base type  #самый главный тип(основа Всех классов)
(END)                  # лишенный характерных черт
>>> #пример про непеременную a (говорят, имя или идентификатор)
>>> a = 5     #ссылка на объект типа int и знач-ем 5
>>> id(a)     #уникальная идентичность -- через встр-ую функцию
139720352475520
>>>
```

- значение объекта (экземпляра класса *int*) равно 5, а его идентичность – адрес в памяти через встроен функцию *id()*.

Лексические термины:

- *Идентификаторы* – это имена, используемые для идентификации переменных, функций, классов, модулей и других объектов (чаще слева от знака =);
- *числовые литералы.* В языке Python существует *четыре* типа *числовых литералов*: Булевы значения, целые, плавающие, комплексные (чаще справа от знака =);
- *строковые литералы* используются для определения последовательностей символов и оформляются как текст (чаще справа от знака =);
- Внутри строковых литералов символ обратного слэша (backslash) (\) используется для экранирования специальных символов.

Сравнение объектов через операторы is и ==

```
>>> aa = [1] # объект aa одноэлементный список
>>> bb = [1] # объект bb одноэлементный список
>>> aa is bb, aa == bb, id(aa) == id(bb), id(aa) is id(bb)
(False, True, False, False) #списки равны только по значениям
>>> сравниваем идентичности, значения и адреса в памяти
>>> у объектов (двух как бы одинаковых списков)
>>> aa, bb  = [1], [71] # теперь разные по значениям объекты
>>> type(aa) is type(bb), type(aa) == type(bb),
(True, True)
>>> type(id(aa)) == type(id(bb)), type(id(aa)) is type(id(bb))
(True, True)
>>> здесь другое, исходим из того, что """Тип объекта
(вызов функции type) возвращает сам бъект,который называется
 (описывается) классом объекта. Этот объект (тип) уникален и
 всегда один и тот же для всех экземпляров данного типа!
 (здесь видно,что лишь бы два списка, поэтому везде true)
"""
```

Функциональность встроенных объектов (чего могут)

- Функциональность определяется методами их классов и у каждого желательно знать хотя бы основные.
- В примере на предыдущем слайде с присваиванием, *a* есть *идентификатор*(имя) *объекта*(тип целый). Список методов у *a* можно получить через встроенную функцию dir(a).
- Всего в списке их будет 70 через len(dir(a)), но из них, так называемые, специальные методы (с двойным подчерком) нам пока не понадобятся, и остаются всего 8.
- Их описание доступно либо через *help(int)*, либо из кода сценария через, так называемое лист комприхеншн, списковое включение (в квадр скобках выдает список из имен 8 методов):

```
>>> [i for i in dir(a) if "__" not in i]
['bit_length', 'conjugate', 'denominator', 'from_bytes',
'imag', 'numerator', 'real', 'to_bytes']
>>>#6 операций с целыми: + - // * ** % обеспечив-щих ЗАМЫКАНИЕ
>>>#смотрим внимательно и запоминаем
```

Как бы “магические числа“ Питона (начиная с 6-й строки это методы встроенных типов, сиречь классов)

33 (всего ключевых слов в языке),
13 (всего встроенных типов (видов типов или классов)),
68 (всего встроенных функций – builtin func),
37 (всего разделов стандартной библиотеки),
11, 83 (специальных атрибутов, специальных методов)
8 (всего методов класса int - целое, см предыд пример с $a = 5$),
7 (всего методов класса float - действительное),
11(всего методов класса list - список),
2 (всего метода класса tuple - кортеж),
44 (всего метода класса str - строка),
17 (всего методов класса set - множество),
11 (всего методов класса dict - словарь),
39 (всего методов класса bytes - байтс) – неизменяемый.
47 (всего методов класса bytearray - байтаррЕй) – изменяемый.
17 (всего методов класса memoryview) – раб-ать с кодом без копиров-я.

Built-in Functions
(также объекты, см. №5,8,13 из слайда 10).
Встроенные функции - их “всего” 68
(некоторые должны знать, подробнее в Лек.4)

```
abs(x)          dict()          help([o])  min(i)          setattr()
all(i)          dir([o])        hex(x)     next(itrtr)     slice()
any(i)          divmod()        id(o)      object()        sorted(i)
ascii(o)        enumerate(i)    input()    oct(x)    staticmethod(f)
bin(x)          eval()          int(x)     open()          str()
bool(x)         exec(o)         isinstance() ord()         sum(i)
bytearray()     filter(f,i)     issubclass() pow(x,y)      super()
bytes()         float(x)        iter(o)    print()         tuple(i)
callable(o)     format()        len()      property()      type()
chr(int)        frozenset(i)    list(i)    range()         vars()
classmethod(f)  getattr()       locals()   repr(o)        zip(*i)
compile()       globals()       map(f,i)   reversed(s)     __import__
complex()       hasattr(o,n)    max(i)     round()
delattr(o,n)    hash(o)    memoryview(o)   set([i])
>>> help(abs)        # вызываем встроенный help по любой их них
Help on built-in function abs in module builtins:
abs(x, /)
Return the absolute value of the argument. # слова и смысл
(END)                                  # понятны, вспоминаем школу
```

Основные инструкции языка (из составных, у кот-ых внутри еще инстр-ии).

Некоторые функции (из встроенных).

Некоторые модули со своими методами

- простые инструкции языка (simple statements, не путать с операторами), и их всего 14:
  expression_stmt, assert_stmt, assignment_stmt, augmented_assignment_stmt, annotated_assignment_stmt, pass_stmt, del_stmt, return_stmt, yield_stmt, raise_stmt, break_stmt, continue_stmt, import_stmt, global_stmt, nonlocal_stmt
- составные инстр-ии (compound statements), всего10:
  if_stmt, while_stmt, for_stmt, try_stmt, with_stmt, funcdef, classdef, async_funcdef, async_for_stmt, async_with_stmt (последние три корутинные инструкции)

- Составные инструкции:(if elif else, for x in ..., while)

```
if boolean_expression1:
  блок1
elif boolean_expression2:
  блок2
elif boolean_expressionN:
  блокN
else:
  else_блок
```

```
for expression in iterable:
  for_блок
else:
  else_блок

while boolean_expression:
  while_блок
else:
  else_блок
```

```
#У while и for условный блок ["else" ":" else_suite] не
#обязателен, но когда стоИт, то сработает, если в главном
#условии фолс, (цикл нормально завершился)
# и если нет внутреннего break
#Вместо того, чтобы всегда выполнять арифм-ую прогрессию
#чисел или давать шаг итерации, и/или условие остановки,
#инструкция for выполняет итерации по элементам любой
#последовательности (список или строка), В ПОРЯДКЕ ИХ
#ПОЯВЛЕНИЯ В ПОСЛЕДОВАТЕЛЬНОСТИ.
```

- if – Условная инструкция

```
>>> x = 15                    # пусть x ссылается на 15
>>> if x % 10 == 0:
...     print("кратно 10")
... elif x % 5 == 0:          # elif и else может не быть
...     print("кратно 5")
... else:
...     pass  #инструкция, кот-я делает ничего(как бы заглушка,
...           #если объявлен else, то у него должно быть тело
>>> кратно 5
```

- Тернарная инструкция

```
>>> "even" if x % 2 == 0 else "odd"  # четное, если x кратно 2
'odd'                                # при заданном x -- нечетно
>>> import sys    # см библ раздел PSL29.1 модуль sys
>>> offset = 20 if sys.platform.startswith("win") else 10
>>> offset  # какая у нас платформа win (windows) или другая
>>> 10    # ну, у меня, конечно, линукс (не win)
```

- while и for инструкции цикла

```
>>> i = 0
>>> while i < 10:
...     i += 1
...
>>> i
10
```

```
>>> sum = 0
>>> for i in range(1, 6):
...     sum += i
...
>>> sum
15
```

- Существуют инструкции *break* и *continue*, похожие на те, что в Си.
- У *while* и *for* условный блок ["*else*:"*suite*] не обязателен
- У составного присваивания, типа += и *= выигрыш по памяти и скорости будет только при работе с изменяемыми объектами и специально со строками.

- встроенные функции *range()* и *reversed()*

```
class range(object)  #Help на класс range в модуле builtins:
|  range(stop) -> range object #поток чисел(идля итерирования)
|  range(start, stop[, step]) -> ранджовый объект,оч.интересный
class reversed(object) #Help на класс reversed в модуле builti
|  reversed(sequence) -> поэлементный реверс итератор посл-ти
range()                  # рантаймное исполнение!!!(без памяти)
# неизм-ая послед-ть, принимает три
# аргумента: начало полуинтервала, конец полуинтервала и
# опциональный шаг, который может быть отрицательным
>>> range(0, 5, 2)                    >>> list(range(0, 5, 2))
range(0, 5, 2)                        [0, 2, 4]
>>> list(range(4, -1, -2))
[4, 2, 0]
>>>
reversed(s)  # перечисляет эл-ты переданной ей
# последовательности в обратном порядке
>>> list(reversed([1, 2, 3]))
[3, 2, 1]
```

- модуль *random* и некот методы (*randint,choice,shuffle,sample*)

```
>>> import random
>>> random.randint(0, 20)    # от рандома рандомное целое
9
>>> random.randint(0, 20)
2       # но, чтобы получить последов-ть таких чисел, надо:
>>> aa = []  #сначала заводим пустой список,чтобыего нарастить
>>> for i in range(6): # повтор 6 раз через ранджовый объект
...     aa.append(random.randint(0, 20))
...
>>> aa
[15, 4, 18, 1, 12, 13]    # получили список из 6 случ целых
#Но лучше через списк-е включение (в духе Питона, идиоматично)
>>> aa = [random.randint(0, 20) for i in range(6)]
>>> aa       # надо именно так (см. Дзен Питона п.13)
[5, 8, 11, 13, 0, 0]
>>> aa = random.sample(range(21), 6) # но тогда не рандомные,
>>> aa                          #а 6 уникальных целых (без повторов)
[17, 5, 14, 16, 11, 20]
```

- выбрать случ элемент из последовательности, перемешать на месте, выбрать указ-ое число случ элт-в из послед-ти

```
>>> random.choice(range(21))
8
>>> random.choice('abcdefghijkl')
'f'
>>> [random.choice(range(21))  for i in range(6)]
[4, 14, 11, 20, 20, 15]  #из20 элем-ого диапазона 6 случайных
>>> aa                       # послед-ть
[15, 4, 18, 1, 12, 13]
>>> random.shuffle(aa)       # перемешиваем послед-ть
>>> aa
[4, 13, 15, 1, 18, 12]
>>>
>>> random.sample(aa, 3)
[18, 15, 13]                 #подпослед-ть из 3 уникальных из 6
>>>
```

Из иерархии типов:

4. numbers.Number (int, bool, float, complex – неизменяемые)
   - 8 (всего методов класса int – целые).
   - Boolean Operations – and, or, not
   - Comparisons (сравнения, сравнительные операторы)

всего методов класса int: *'bit_length'*, *'conjugate'*, *'imag'*, *'to_bytes'*, *'numerator'*, *'denominator'*, *'from_bytes'*, *'real'*

```
>>> a = 9
>>> a.bit_length()    # сколько двоичных битов в 9
4
>>> a = -9
>>> bin(a)
'-0b1001'
>>> a.bit_length()
4
>>> (255).to_bytes(2, byteorder='big') # двухбайтовое
b'\x00\xff'                            # представление числа
>>> (255).to_bytes(10, byteorder='big')  # для "остроконечных"
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\xff'
>>> aa = 4
>>> aa.denominator                  >>> aa.numerator
1      # знаменатель               4     # числитель
# у рациональных                  # у рациональных
```

Булевы – подтип целых.True и False два единств-х экземпляра класса bool. Boolean Operations: *and*, *or*, *not*.

bool(x) returns True when the argument x is true, False otherwise.

```
>>> a, b, c = 9, 3, 0
>>> a and b, b and a, a and c
(3, 9, 0)                  # в ответе не булево значение, а операнд,
>>> a or b, b or a, a or c, c or a      # вычисленный в
(9, 3, 9, 9)                            # булевом контексте
>>> a and b, b and a, a and c, c and a
(3, 9, 0, 0)
>>> not a, not c
(False, True)
>>> qq1 = 22
>>> qq2 = 1
>>> qq1 or qq2    # ленивые вычисления(у or), если qq1 не ноль,
22                 # то ответ qq1
>>> qq1 and qq2  # если qq1 и qq2 не ноль, то ответ qq2
1
```

- внимание!! Как оценить по булевски признак окончания/(исчерпывания) всех этих последовательностных, сЕтовых, мАпинговых коллекций при циклах или сравнениях?
- при while или if условиях, окончание обхода/(сравнения) любой коллекции должно давать False. И в языке за False принимаются пустые коллекции строк, кортежей, списков, множеств и словарей (””, (), [], set(), {}). Хотя при прямых проверках это не проходит!!
- само собой, интОвые, флОатовские, комплексные нули также дают False, (0, 0.0, 0j). Их прямые проверки работают.

```
>>> ll = [1, 2]
>>> while ll.pop(): print('ok')
...
ok
ok
IndexError: pop from empty list
>>>#список исчерпался, сталFalse ok
```

```
>>>#но прямые проверки не идут
>>> if [] == False: print('ok')
... # здесь "не хочет"
>>> # но по-другому работает
>>> if [] != True: print('ok')
...
```

Comparisons (сравнительные операторы)

Операция	Значение
`<`	строго less than
`<=`	less than or equal
`>`	строго greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity (идентичность объекта)
`is not`	negated object identity (отсутствие идентич.)

```
>>> a = 9
>>> 0 < a < 10   # меньшее слева, и не надо 10 > a > 0.
True
```

Из иерархии типов:

5. Sequences (упоряд-енная коллекция эл-тов с неотриц-ми индексами, с длиной и срезами. Неизменяемые: strings, tuples, bytes, ranges. Изменяемые: lists, byte arrays.)
   - коллекция – список

     (всего методов класса list 11)

- список (list) – это упорядоченная последовательность из нуля или более ссылок на объекты, контейнерная последовательность[†]
- функционал списка как объекта (без спец методов):
  1. добавить в конец списка
  2. очистить все элементы
  3. копирование списка (поверхностное)
  4. сколько раз встречается то или иное значение
  5. расширить (поэлементно)
  6. задав значение, узнать его индекс (первый)
  7. вставить объект перед указанным индексом
  8. выкинуть элемент, указав его индекс, по дефолту последний
  9. удалить указанный эл-т
  10. перевернуть на месте
  11. отсортировать на месте
- список – самый фундаментальный тип послед-ти, хранит объекты разных типов. Подробнее, см. *help(list)*

[†]На примере списка вспомните 4 части определения последовательности

```
>>> ml = []       # пустой список или можно еще через list()
>>> [1] * 4       # умножаем на целое
[1, 1, 1, 1]
>>> [2, 5] + [2, 4]    # конкатенация
[2, 5, 2, 4]
>>> x = [2, [3, (55,), -2.]]
>>> x
[2, [3, (55,), -2.0]]    # вложенные списки
>>> len(x)
2                        # почему только 2?
>>> x2 = [3, -4, 22]
>>> x2[1] = 0
>>> x2
[3, 0, 22]               # проверили на изменяемость
>>> x = [0, 1]           # проверка на множеств-ое присваив-ие
>>> i = 0
>>> i, x[i] = 1, 2       # i is updated, then x[i] is updated
>>> print(x)
[0, 2]                   # i изменился и x от i соответно
```

Оператор * – оператор распаковки последовательности

```
#!!! Для извлечения 2-х и более частей списка с разных сторон
>>> first, *rest = [8, 3, 5, 7]
>>> first
8
>>> rest    # выражение *rest -- выражение со звёздочкой
[3, 5, 7]
>>> first, *mid, last = "АлександрI, НиколайI, АлександрII,
АлександрIII, НиколайII".split() #расщепляем строку по пробелу
>>> first, mid, last  #теперь смотрим уже с распаковкой mid-а
('АлександрI,', ['НиколайI,', 'АлександрII,','АлександрIII,'],
'НиколайII')    # от сплитА внутри кортежа список
>>> *dir, exec = "/usr/bin/vim".split('/') #расщепляем по слэш
>>> dir, exec  # dir распаковался
(['', 'usr', 'bin'], 'vim')
>>>
```

Создание длинных списков программным способом (списковое включение, см. висок-ый год в лек1)

```
>>> lm = [i for i in dir(list) if "__" not in i]
>>> lm #список создаётся налету, а не делается заранее
['append', 'clear', 'copy', 'count', 'extend', 'index',
'insert', 'pop', 'remove', 'reverse', 'sort'] #те самые 11 шт.
>>> x2.append(33)
>>> x2
[3, 0, 22, 33]
>>> x2.index(22)
2



>>> x2.count(0)
2
```

```
>>> x2.pop()
33
>>> x2
[3, 0, 22]
>>>x2.insert(0, 0)
>>> x2
[0, 3, 0, 22]
>>> del x2[0]
>>> x2
[3, 0, 22]
```

Слайсы (slice – вырезаем подиндексы с начала и с конца)

```
class slice(object) #Help on class slice in module builtins:
|  slice(stop)
|  slice(start, stop[, step])
|  Create a slice object.  # слайс объект(см №13 на слайде 10)
>>> x = list(range(1, 7))
>>> x
[1, 2, 3, 4, 5, 6]
>>> x[:2]                    >>> x[2:]
[1, 2]                       [4, 5, 6]
>>> x[1:3]
[2, 4]
>>> x[5:0:-1]                 >>> x[5:0:-2]
[6, 5, 4, 2]                 [6, 4]
>>> x2 = x[:-1]               >>> x2 = x[::-1]
>>> x2                        >>> x2
[1, 2, 4, 5]                 [6, 5, 4, 2, 1]
```

Слайсы присваиваются и удаляются

```
>>> x2 = [6, 5, 4, 3, 2, 1]
>>> x2[1:3] = [10, 20, 30]  # вместо 5 и 4 вставляем 3 эл-та
>>> x2
[6, 10, 20, 30, 3, 2, 1]
>>> x2[1:1]                          >>> x2[1:2]
[]                                   [10]
>>> x2[1:2] = [77, 55]
>>> x2
[6, 77, 55, 20, 30, 3, 2, 1]
>>> x2[1:1] = [47, 45]
>>> x2
[6, 47, 45, 77, 55, 20, 30, 3, 2, 1]
>>> del x2[5:7]
>>> x2
[6, 47, 45, 77, 55, 3, 2, 1]  >>> x2[4::2] = [11, 22]
>>> x2  # вставляем два и место должно быть только под два
[6, 47, 45, 77, 11, 3, 22, 1] # иначе ошибка
```

- Индексы с начала и с конца

```
>>> x = [1, 2, 4, 5, 6] #индекс послед-го -1, предпосл-го -2...
>>> x[-2]
5
>>> x[len(x) -1]
6
>>> x[5]
IndexError: list index out of range # нет такого номера
```

- реверс и сорт изменяют на месте и не создают новых списков. Правило такое - если изменяет на месте, то возвращает None

```
>>> x = [1, 4, 2, 6, 5]
>>> x.reverse()
>>> x
[5, 6, 2, 4, 1]
>>> id(x)
139958300169096
```

```
>>> id(x)
139958300169096
>>> x.sort()
x
[1, 2, 4, 5, 6]
>>> id(x)
139958300169096
```

- метод сорт не любит разнотипность, понятно, но!!

```
>>> ll = [1, 9, '19', 2, '022']
>>> ll.sort()
TypeError: '<' not supported between instances of 'str'
and 'int'
```

- но во встроенной функции sorted() имеется необязательный аргумент key и он сильно помогает. Тогда разнотипные списки можно сортировать если его "числа" похожи на строки.

```
>>> sorted(ll, key=int)
[1, 2, 9, '19', '022']
>>> sorted(ll, key=str)
['022', 1, '19', 2, 9]
>>> ll
[1, 9, '19', 2, '022']
>>>
```

append и pop

```
>>> m_l = [1, 3, 5, 7]
>>> m_l.append(0)
>>> m_l.append('77')
>>> m_l
[1, 3, 5, 7, 0, '77']    #видим добавились сначала 0 потом'77'

# pop без аргумента удаляет последний!!!!
>>> m_l.pop(4)
0                        #здесь достаем 4-й эл-т
>>> m_l
[1, 3, 5, '77']
>>> m_l.pop()
'77'                     # а здесь достаем последний
>>> m_l
[1, 3, 5]
>>>
```

Работа с матрицами

```
>>> mat = [
... [1, 2, 3],
... [4, 5, 6],
... [7, 8, 9]
... ]
>>> mat
[[1, 2, 3], [4, 5, 6], [7, 8, 9]] # 3-х элементный список
>>> for row in mat:    # печатается индекс
...     print (row)
...
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
>>> mat[0], mat[1], mat[2]
([1, 2, 3], [4, 5, 6], [7, 8, 9])
>>>
```

Транспонировать матрицу через списковое включение с двумя for:

```
>>> mat_trans = [[row[i] for row in mat] for i in [0, 1, 2]]
>>> mat_trans    # читать справа налево, i справа -- строки, a
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]    # row слева -- столбцы
>>> for i in [0, 1, 2]:  # то же самое, но многословно
...     for row in mat:
...         print(row[i], end=" ")
...     print()
...
1 4 7
2 5 8
3 6 9
# Для создания двумерных массивов использовать вложенные
# списковые включения, разместив один, являющегося строкой,
# внутри другого
```

Транспонировать матрицу(2) через встр функцию zip с распаковкой итераб-ых аргументов (через *), но мелкий минус.

- у функции zip аргументом стоИт один или несколько (через ,) итерабельных объектов;
- на выходе кортежи, с соотв-щими эл-тами этих итер-ых объектов;
- zip – генератор, ничего не выводит, надо напр-р, "о-списковать".

```
>>> aa1 = ['ab']
>>> list(zip(*aa1))  # строка итерируется по a и b
[('a',), ('b',)]     # список от генератора ("о-списковали")
>>> aa = ['abc', 'defg', 'hjik'] #слепливаются толькоСоотв-щие
>>> list(zip(*aa))  # строка итерируется по элементам списка
[('a', 'd', 'h'), ('b', 'e', 'j'), ('c', 'f', 'i')]
>>> # тогда матрицу как двумерный список можно зиповать и
>>> # таким образом транспонировать
>>> list(zip(*mat))   # внутр кортежи можно сразу в список
[(1, 4, 7), (2, 5, 8), (3, 6, 9)] # через списк включ-е
>>> [list(i) for i in list(zip(*mat))]
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]  # вот финал
>>>
```

Работа с матрицами2

```
>>> n, m = 5, 6                    # 5 строк и 6 столбцов
>>> [ [0] * m for i in range(n)]
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
>>># Теперь, сложнее, - внутренний список также можно создать
>>># при помощи, например, такого включателя:
>>># [0 for j in range(m)]. Вложив один включатель в другой
>>>#получим вложенные списковые включения
>>> [ [0 for j in range(m)] for i in range(n)]
[[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0]]
>>># Например, пусть нужно задать следующий массив (для удобсте
>>># добавлены дополнительные пробелы между элементами):
0  0  0  0  0  0
0  1  2  3  4  5
0  2  4  6  8 10
0  3  6  9 12 15
0  4  8 12 16 20
```

```
# В этом массиве n = 5 строк, m = 6 столбцов,
# и элемент в строке i и столбце j вычисляется по формуле:
# A[i][j] = i * j.
>>> mat2 = [[ i * j for j in range(m)] for i in range(n)]
>>> mat2
[[0, 0, 0, 0, 0, 0],
[0, 1, 2, 3, 4, 5],
[0, 2, 4, 6, 8, 10],
[0, 3, 6, 9, 12, 15],
[0, 4, 8, 12, 16, 20]]
>>>
```

Итак! (выводы)

- В классической оболочке Питона имеется достаточно развитый встроенный *help()* для онлайн документации, а оболочке *ipython* еще, кроме прочего, подсветка синтаксиса и легкие сниппеты;
- В Питоне объекты – наше "фсё", а встроенных типов (классов) для них аж 13;
- Методы встроенных классов это функции, выполняющие разнообразные операции над объектом с вызовом через точку;
- Можно сказать, что последовательности в Питоне это кладесь возможностей (в смысле количества и разнообразия операций с этими типами, причем как встроенными так и бибилиотечными), но это больше в след лекции. Тут же отметим возможность однотипности операций, таких как: итерирование, получение среза, сортировка, конкатенация и др.

Финиш Лек 2

the End по Лекции 2, вопросы?

Далее

Вопросы(для экзамена)/упражнения(для лабораторных)

14. Питон "любит" конструкции, чтобы не резервировать заранее память, а создавать и вычислять налету (рантайм). Назовите хотя бы одну такую?
15. имеется два списка, назовите 2 способа приклеить к первому второй?
16. в списке сколько-то элементов, назовите три способа удаления последнего и не обязательно только через методы списка?
17. задать список можно тремя способами, перечислите их, плииз?
18. стэк, работающий по принципу LIFO (last in first out) реализуется в списке с помощью 2-х методов – каких?
19. если наращивать список с помощью append() или extend() методов, то можно ошибиться, объясните или приведите пример?
20. три метода наращивания списка, примеры? (разумеется в хвост)
21. имеется список ['mama', 'myla', 'ramu']. Как проверить сколько в нем элементов и есть ли там какой-то конкретный в наличии?

20. продемонстрируйте, что при копировании список2 = список1 эти имена будут ссылаться всё равно на один и тот же объект (через какую-то встроенную функцию) и, если мы изменим список1, то автоматом поменяется и список2. Но главное, покажите, что будет, если копировать списки через слайсы, единственность объекта сохраниться?
21. в Питоне при применении булевских операций к числам ответы вовсе не булевские, по каким принципам формулируются ответы?, что будет в ответе: 5 and 10, 10 and 9, 9 and 0 или 5 or 10, 10 or 9, 9 or 0?, а 0 or 9?
22. Назовите самый быстрый (лёгкий) способ объявления большого списка на 1000 одинаковых элементов, например 1000 чисел 100500?
23. значки + и * плюсуют и умножают что?, приведите примеры?
24. замените (промоделируйте) append для списка , типа
    »> aa = [1, 2]
    »> aa[len(aa):] [] »>aa[len(aa):] = [33] »>aa
    [1, 2, 33]
    и как через лен вставить в начало списка?

25. какими 2-я методами можно работать со списком как со стеком? иначе, как слелать LIFO?
26. какими методами и с чем можно работать с дабл-очередью в отличии от списка
    from collections import deque
    queue.append("Graham")
    queue.popleft()
27. если range() возвращает ренджевый объект, то покажите, есть ли у этого объекта индекс, слайсы или чего еще