Шпаргалка Python 3

Перевод : Макеев Артём @Art_IP
https://github.com/etpu/Python-Memo

Базовые Типы

целые, с точкой, логич., строки...

```
int 783  0 -192  0b010 0o642 0xF3
                 ноль  двоичный 8-чный 16-чный
float 9.23 0.0  -1.7e-6
                        ×10^-6
bool True  False
str "Три\nДва"   Многострочная строка:
     перевод строки  """X\tY\tZ
      'I\'m'         1\t2\t3"""
     экранирование '  симв. табуляции
bytes b"toto\xfe\775"
        16-ричные 8-ричные
```

неизменяемые

Контейнерные Типы

▪ **упоряд-ая послед-сть,** быстрый доступ по индексу, повторяемые

```
list [1, 5, 9]  ["x", 11, 8.9]   ["mot"]   []
tuple (1, 5, 9)  11, "y", 7.4    ("mot",)  ()
```

Не изменяемые (имутабельные) — *выражение только с запятыми →* `tuple`

```
str bytes   (упорядоченная последовательность символов / байт)   "" b""
```

▪ **Контейнеры с ключами**, быстрый доступ по ключу, ключи уникальны

```
словарь dict {"ключ":"значение"} dict(a=3,b=4,k="v")   {}
(ключ/значение)  {1:"one",3:"three",2:"two",3.14:"π"}
множество set {"key1","key2"}  {1,9,3,0}  (не упорядоченные)  set()
```

ключи=хэшируемые знач.(базовые типы, не изм.) `Frozenset` *- не изменяемый set* *пусто*

Идентификаторы

Имена для переменных, функций, модулей, классов

`a...zA...Z_` затем `a...zA...Z_0...9`

- кирилица разрешена, но не желательна
- ключевые слова запрещены
- мал/БОЛ буквы различаются (hey != Hey)

☺ `a toto x7 y_max BigOne`
☹ ~~`8y and for`~~

Присвоения Переменным

`=` присвоение ⇔ **привязка** *имени к значению*

1) рассчитать выражение справа
2) присвоить по порядку именам слева

```
x=1.2+8+sin(y)
a=b=c=0   присвоение одинакового значения
y,z,r=9.2,-7.6,0   множеств. присв.
a,b=b,a   поменять местами значения
a,*b=seq  } распаковать последов-сть в
*a,b=seq  } переменную и список
```

так же :

```
x+=3  инкремент ⇔ x=x+3   *=
x-=2  декремент ⇔ x=x-2   /=
x=None «undefined» константа   %=
del x  Удалить переменную x   …
```

Преобразование

`type(`*выражение*`)`

```
int("15") → 15
int("3f",16) → 63      вторым параметром указываем систему счисления
Int(15.56) → 15        отбрасываем дробную часть
float("-11.24e8") → -1124000000.0
Round(15.56,1)→15.6    округляем до 1 знака после запятой
bool(x) False когда x=0, контейнер x пустой, x None или False; True в др. случ.
str(x)→"..."  строковое представление x (см. Форматирование на след. листе)
chr(64)→'@'  ord('@')→64   Код символа ↔ Символ
repr(x)→"..."  формальное строковое представление x
bytes([72,9,64]) → b'H\t@'
list("абв") → ['а','б','в']
dict([(3,"three"),(1,"one")]) → {1:'one',3:'three'}
set(["one","two"]) → {'one','two'}
разделитель  последовательность str  → объединенная str
':'.join(['toto','12','pswd']) → 'toto:12:pswd'
str разделенная пробелами  → list из str
"слова с пробелами".split() → ['слова','с','пробелами']
str с разделителем  str  → list из str
"1,4,8,2".split(",") → ['1','4','8','2']
Последовательность одного типа → list др. типа (с помощью Спискового включения)
[int(x) for x in ('1','29','-3')] → [1,29,-3]
```

Индексирование Последовательностей

для списков, кортежей, строк, байт…

```
отрицат. индекс  -5  -4  -3  -2  -1
положит. индекс   0   1   2   3   4
lst=[10, 20, 30, 40, 50]
положит. срез     0   1   2   3   4   5
отрицат. срез    -5  -4  -3  -2  -1
```

Количество элементов

`len(lst)→5`

Счет с **0**
(тут от 0 до 4)

Доступ к **элементу** через `lst[`*index*`]`

```
lst[0]→10  ⇒ первый      lst[1]→20
lst[-1]→50 ⇒ последний   lst[-2]→40
```

В изменяемой последовательности (`list`), можно удалять элементы с помощью `del lst[3]` *и изменять элементы* `lst[4]=25`

Доступ к части последовательности через `lst[`*начало среза*`:`*конец среза*`:`*шаг*`]`

```
lst[:-1]→[10,20,30,40]   lst[::-1]→[50,40,30,20,10]   lst[1:3]→[20,30]   lst[:3]→[10,20,30]
lst[1:-1]→[20,30,40]     lst[::-2]→[50,30,10]         lst[-3:-1]→[30,40]  lst[3:]→[40,50]
lst[::2]→[10,30,50]      lst[:]→[10,20,30,40,50]  теневая копия последовательности
```

Для изменяемых последовательностей (`list`), удаление через `del lst[3:5]` *и изменение через присвоение* `lst[1:4]=[15,25]`

Булева Логика

Сравнения : `< > <= >= == !=`
(результат: булевое знч.) ≤ ≥ = ≠

`a and b` логическое И
Верно когда оба значения истина

`a or b` логическое ИЛИ
Верно когды любое значение истина

`and` / `or` *возвращают значение одной из переменных* `a` *или* `b`*, даже если они не содержат логических значений.*

`not a` Логическое нет

`True` константа « Истина »
`False` константа « Ложь »

Блоки Инструкций

Родительская инструкция :
→ *Блок инструкций 1…*
⋮
Родительская инструкция :
→ *Блок инструкций 2…*
⋮
след. инструкция после блока 1

Отступы !

настройте редактор или IDE для вставки 4 пробелов вместо Tab

Импорт Модулей и Имён

модуль `truc` ⇔ *файл* `truc.py`

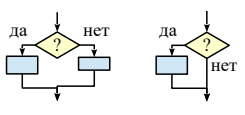
```
from mymod import mod1,mod2 as mod
```
→*прямой доступ к имени,* `as` *- переименовывает*
```
import mymod  →обращение через mymod.mod1 …
```

Модули и пакеты ищутся в *python path* (см. `sys.path`)

Условный Оператор

Выражение в блоке выполняется Только если

`if` *Логическое Условие* `:`
→ *Блок инструкций*



Может быть несколько условий *elif*, но может содержать только одно последнее условие else. Выполняется только первый блок истинного условия.

```
if bool(x)==True:  ⇔ if x:
if bool(x)==False: ⇔ if not x:
```

```
if age<=18:
  state="Kid"
elif age>65:
  state="Retired"
else:
  state="Active"
```

Математика

числа с плавающей точкой… приближенные *Углы в радиусах*

Операторы: `+ - * / // % **`
Приоритеты `(…)` × ÷ ↑ ↑ a^b
целочисленное ÷ ÷ с остатком
`@` → matrix × *python3.5+*`numpy`

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
pow(4,3)→64.0
```

обычный порядок операций

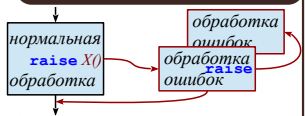
```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
sqrt(81)→9.0   √
log(e**2)→2.0
ceil(12.5)→13
floor(12.5)→12
```

Модули `math`, `statistics`, `random`, `decimal`, `fractions`, `numpy`, и тд.*(см.doc)*

Обработка Исключений

Вызов исключения:
`raise` *ExcClass(…)*

Обработка ошибок:
`try:`
→ *основные инструкции*
`except` *Exception* `as e:`
→ *блок обработки ошибок*



`finally` *блок для финальной обработки. Всегда запускается !*

Цикл с Условием

блок инструкций будет выполняться ***так долго*** *пока условие истинно (true)*

```
while логич. выражение:
    Блок инструкций
```


```
s = 0      инициализация перед циклом
i = 1      условие с хотя бы 1 значением переменной (в данном случае i)
while i <= 100:
    s = s + i**2
    i = i + 1
print("sum:",s)
```

Обязательно необходимо изменить Значение переменной, иначе мы получим бесконечный цикл

$$S = \sum_{i=1}^{i=100} i^2$$

Бесконечные циклы опасны!

Управление циклом

`break` *остановить и выйти*
`continue` *след. итерация*
`else` *блок для* ***нормального*** *выхода из цикла.*

Вывод

```
print("v=",3,"cm :",x,",",y+4)
```

элементы для отображения: литералы, переменные, выражения

параметры `print`:

- `sep=" "` разделитель, по умолчанию пробел
- `end="\n"` конец строки, по умолчанию новая строка
- `file=sys.stdout` печать в файл, по умолч. стандартный вывод

Ввод

```
s = input("Инструкции:")
```

`input` always returns a **string**, convert it to required type (cf. boxed *Conversions* on the other side).

Операции с Контейнерами

`len(c)` → количество эдементов
`min(c)` `max(c)` `sum(c)`
`sorted(c)` →`list` отсортированая копия

Прим: для словарей и множеств операции работают с ***ключами***

`val in c` → логич., содержится `in` (отсутствует `not in`)
`enumerate(c)` → *итератор* парами (ключ, значение)
`zip(c1,c2...)` → *итератор* кортежей c_i соединяет элементы по индексу
`all(c)` → `True` если **все** `c` элементы истина, иначе `False`
`any(c)` → `True` если **хотя бы 1** элемент `c` истина, иначе `False`

Только для ***последоват-стей с порядком*** *(списки, кортежи, строки, байты…)*

`reversed(c)` → *инвертир-ть* `c*5`→ повторить 5 раз `c+c2`→ соединить
`c.index(val)` → *позиция* `c.count(val)` → *подсчет вхождений*

```
import copy
copy.copy(c) → поверхностное копирование
copy.deepcopy(c) → глубокое копирование
```

Операции со Списками

изменяют исходный список

`lst.append(val)`	добавить элемент *val* в конец
`lst.extend(seq)`	добавить последовательность *seq* в конец
`lst.insert(idx,val)`	вставить значение *val* по индексу *idx*
`lst.remove(val)`	удалить первое вхождение *val*
`lst.pop([idx])` →*знач.*	удалить и вернуть значение по индексу *idx*
`lst.sort()` `lst.reverse()`	сортировать / обратить список

Операции со Словарями

`d[key]=значение` `d.clear()`
`d[key]` → *значение* `del d[key]`
`d.update(d2)` обновить/добавить пары
`d.keys()` `d.values()` `d.items()` →*просмотр ключей/знач./пар*
`d.pop(key[,default])` → *значение*
`d.popitem()` → *(ключ,значение)*
`d.get(key[,default])` → *значение*
`d.setdefault(key[,default])` →*знач.*

Операции с Множествами

Операции:

- `|` → объединение (верт-ая черта)
- `&` → пересечение
- `-` `^` → разность/симетричн. разн.
- `<` `<=` `>` `>=` →отношения включ-я

Операторы также сущ-т как методы

`s.update(s2)` `s.copy()`
`s.add(key)` `s.remove(key)`
`s.discard(key)` `s.clear()`
`s.pop()`

Файлы

Сохранение и считывание данных с диска

```
f = open("file.txt","w",encoding="utf8")
```

Файловая **переменная** для операций

имя файла На диске (+путь…)

режим работы
- `'r'` чтение
- `'w'` запись
- `'a'` добавление
- …`'+'` `'x'` `'b'` `'t'`

Кодировка символов в *текстовых файлах*: `utf8` `ascii` `cp1251` …

см. модули `os`, `os.path` и `pathlib`

запись

```
f.write("Привет")
f.writelines(list of lines)
```

читать пуст. строку, при конце файла **чтение**

`f.read([n])` → *кол-во символов*
если n не указано, прочитать все до конца!
`f.readlines([n])` → *список из n строк*
`f.readline()` → *следующая строка*

текстовый режим `t` *по умолч. (read/write* `str`*), возможен бинарный режим* `b` *(read/write* `bytes`*).* ***Конвертируйте из/в нужный тип!***

`f.close()` и не забывать **закрывать файл** после использования !

`f.flush()` очистить буфер `f.truncate([size])` обрезать файл

прогресс считывания/записи в файле, изменяется с помощью:

`f.tell()` →*позиция* `f.seek(position[,origin])`

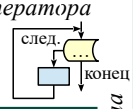
Очень часто: используют **with** для гарантированного закрытия файла и прочтения строк файла через цикл **for**:

```
with open(...) as f:
    for line in f :
        # обработка строки
```

Цикл с Перебором Значений Посл.

Блок инструкций будет выполнятся для всех элементов контейнера или итератора

```
for var in послед-сть:
    Блок инструкций
```



Проход по **элементам** последовательности

```
s = "Мой Текст"   инициализация перед циклом
cnt = 0
```

Переменная цикла, ***значение управляется циклом for***

```
for c in s:
    if c == "e":
        cnt = cnt + 1
print("found",cnt,"'e'")
```

Подсчитывает количество букв `e` *в строке*

цикл по dict/set ⇔ цикл по ключам в последовательности
используйте *срезы* для прохода по последовательностям

Проход по **индексам** последовательностей

- изменить элемент по индексу
- доступ к соседним элементам (до / после)

```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:",lst,"-lost:",lost)
```

Ограничивает значения больше 15 в списке, сохраняя потерянные значения в новом списке

хорошая привычка: не изменять переменную цикла

Проход одновременно по **индексам** и **значениям**:

```
for idx,val in enumerate(lst):
```

Числовые последоват-сти

```
range([start,] end [,step])
```

`start` по умолч. 0, *end* не включается, *step* (шаг), по умолч. 1

`range(5)` → 0 1 2 3 4 `range(2,12,3)` → 2 5 8 11
`range(3,8)` → 3 4 5 6 7 `range(20,5,-5)` → 20 15 10
`range(len(seq))` → *последовательность из индексов seq*

`range` *дает неизм. послед-сть чисел, выдавая по мере необход-ти*

Определение Функций

имя_функции (идентификатор) параметры

```
def do_it(x,y,z):
    """Строка Документации"""
    # Блок инструкций, расчеты, и т.д.
    return res
```

Результат вызова, если нет возврата то вернет `return None`

параметры и все переменные в этом блоке существуют только во время вызова функции

Продвинут.: `def fct(x,y,z,*args,a=3,b=5,**kwargs):`
**args - позиционные аргументы (→*`tuple`*), знач.по умолчанию,*
***kwargs - именованные аргументы (→*`dict`*)*

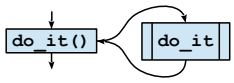
Вызов Функций

```
r = do_it(3,i+2,2*i)
```

хранение/испол-ние возвращаемого знач. *один аргумент для каждого параметра*

используя имя функции с круглыми скобками – вызывает эту функцию

Продвинут.:
**послед-сть*
***словарь*



Строковые Операции

`s.startswith(prefix[,start[,end]])` `s.endswith(suffix[,start[,end]])` `s.strip([chars])`
`s.count(sub[,start[,end]])` `s.partition(sep)` → *(до,разд-ь,после)*
`s.index(sub[,start[,end]])` `s.find(sub[,start[,end]])`
`s.is...()` *тесты по категориям (пример.* `s.isalpha()`*)*
`s.upper()` `s.lower()` `s.title()` `s.swapcase()`
`s.casefold()` `s.capitalize()` `s.center(width,fill)`
`s.ljust(width,fill)` `s.rjust(width,fill)` `s.zfill(width)`
`s.encode(encoding)` `s.split(sep)` `s.join(seq)`

Форматирование

директивы форматирования знач.для формат-я

```
"модель{} {} {}".format(x,y,r) → str
"{селектор:формат!преобразование}"
```

- **Селекторы** :
  - `2`
  - `nom`
  - `0.nom`
  - `4[key]`
  - `0[2]`
- **Формат** :

Примеры:

```
"{:+2.3f}".format(45.72793)
→'+45.728'
"{1:>10s}".format(8,"toto")
→'      toto'
"{x!r}".format(x="I'm")
→'"I\'m"'
```

заполнение *выравнивание* *знак* *мин.ширина* *тип*

`<` `>` `^` `=` `+` `-` *пробел* `0` в начале для заполнения 0

целые: `b` бинарный, `c` симв., `d` десятичн. (по умолч.), `o` 8-ричн, `x` или `X` 16-ричн., float: `e` или `E` экспонен-ая, `f` или `F` фикс.точка, `g` или `G` appropriate (default), строки: `s` , `%` перевод долей

- **Преобразование**: `s` (читаемый текст) или `r` (буквальное преставл-е)