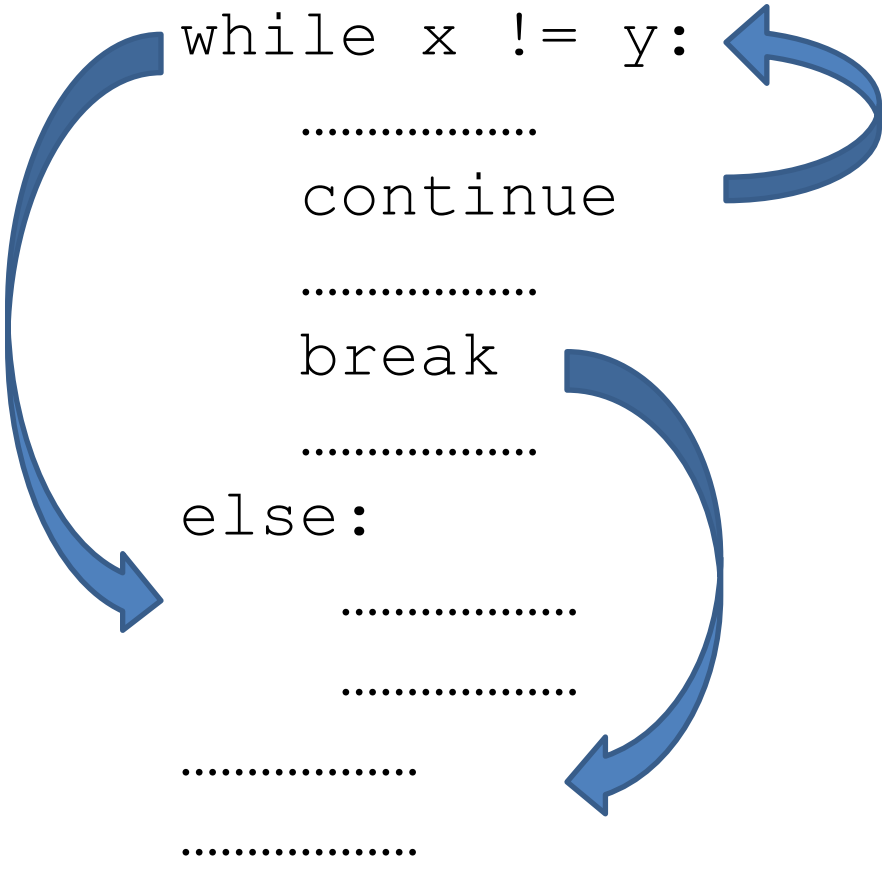
Python3 Cheat Sheet

Управляющие конструкции

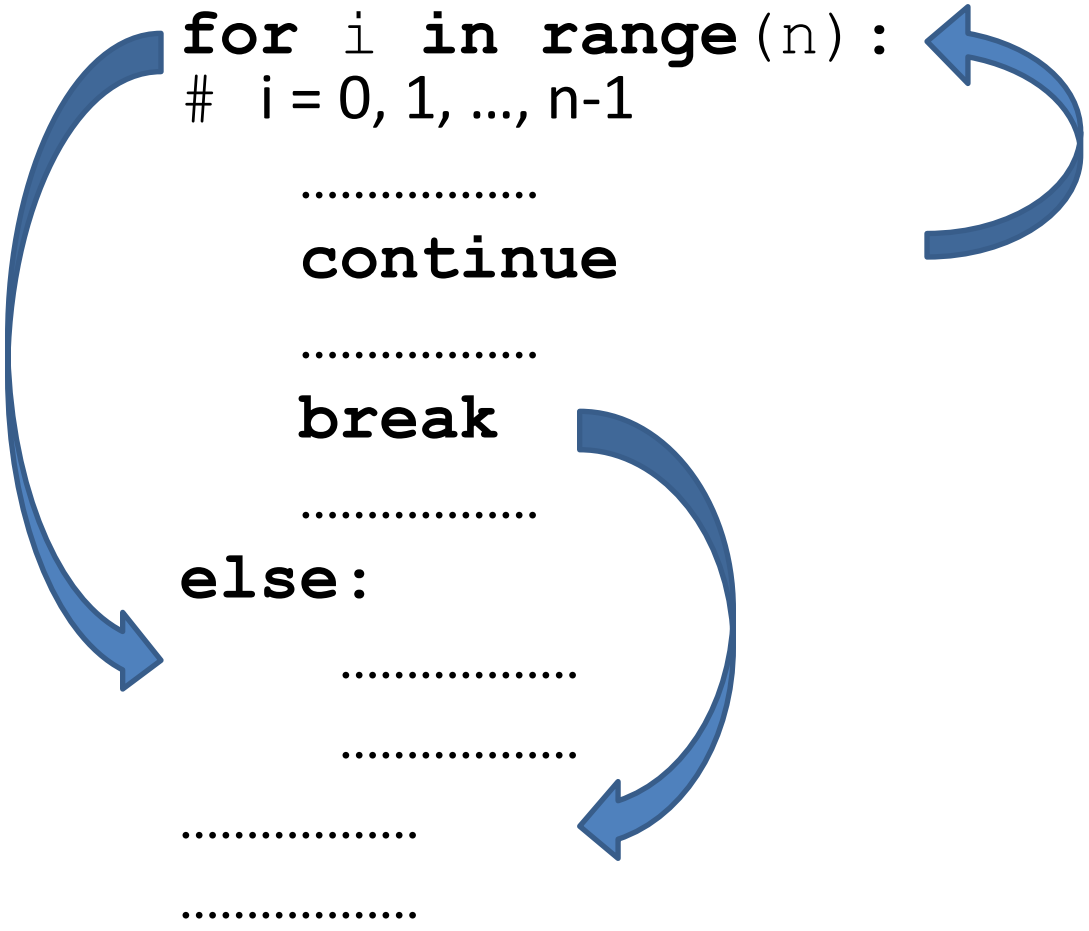
Условный оператор

```
if x > 0:
     ……
     ……
elif x < -1:
     ……
     ……
elif x < 0:
     ……
else:
     ……
……
```

Цикл с предусловием



Цикл for



```
for item in mylist:
#  item = mylist[0], mylist[1], …
        item = 0
        # mylist не меняется!
```

Стандартные типы данных

Числа

Целые числа

Вещественные числа

Комплексные числа

```
>>> z = 2.3 + 3j
>>> z.real, z.imag
(2.3, 3.0)
```

Хранится пара чисел типа float

Тип int

- Размер чисел ограничен только доступной памятью

- >>> 12345 # десятичная запись

12345

>>> 0b1001 #двоичная запись

9

>>> 0xFA # 16-ричная запись

250

- Арифметические операции: +, -, *, /, //, %, **, abs(x)
- Битовые операции: &, |, ~, <<, >>, ^ (xor)

Создание переменных типа **int**

```
a = 17
b = int(17)
c = int('17')
d = int(17.1)
e = round(17.9) # 18
```

Системы счисления

Синтаксис	Описание
bin(n)	Превращает число n в строку, содержащую двоичное представление n: bin(5) = '0b101‘
hex(i)	Превращает число n в строку, содержащую 16-ричное представление n: hex(17) = '0x11'
int(s, [base])	Преобразует строку s в число, предполагая, что оно было записано в системе счисления с основание base (от 2 до 36, по умолчанию 10): int('A2', 11) = 112

Модуль math

- gcd (для целых чисел);
- floor, ceil, trunc;
- sin, cos, …, asin, acos, …;
- log
- pi

Разное

- `# это комментарий; многострочных`
  `# комментариев НЕТ`
- Заглавные и строчные буквы - различаются
- `while x > 0:`

  `pass` # может быть позже здесь будет код
- `None`

Логический тип bool

- Два значения: True и False (с большой буквы)
- False: 0; 0.0; пустая строка, пустой список, …
- True: все остальное
- Можно (но не нужно!) вместо True/False использовать 1/0

Логические операции

- not: возвращает True или False
- and, or : возвращают аргумент, определяющий значение выражения:

```
>>> 'money' or 'life'
'money'
>>> True and 5
5
```

Если второй аргумент не нужен, он не вычисляется:

```
if 2 > 1 and 1/0 = 5:
```

- Скобки не нужны: приоритет логических операций минимальный
- `x < y < z` (эквивалентно `x < y and y < z`)

Числа с плавающей точкой: float

Числа двойной точности (double в C), точность зависит от используемого для сборки Python компилятора C.

Подробнее см. `sys.float_info`:

`dig, epsilon, min, max,…`

decimal.Decimal

- Гарантируют заданную пользователем точность в *десятичной системе*
- *В том числе для чисел типа 0.1, не представимых в виде конечных двоичных дробей*
- *По умолчанию точность – 28 знаков*
- *Удобен для финансовых вычислений*

Decimal: пример

```
>>> 0.1+0.1+0.1-0.3
5.551115123125783e-17

>>> from decimal import Decimal
>>> Decimal('0.1')+Decimal('0.1')+
Decimal('0.1')-Decimal('0.3')
Decimal('0.0')
```

fractions.Fraction

```
>>> from fractions import Fraction
>>> print(Fraction(4,5))
4/5
>>> print(Fraction(4,5)+Fraction(1,2))
13/10
```

Дополнительным модули

- cmath (стандартный)

аналог math для работы с комплексными числами

- NumPy

Научные и инженерные вычисления: эффективная реализация многомерных массивов, линейная алгебра, преобразования Фурье, ...

- SciPy = NumPy + …

статистические функции, обработка сигналов и изображений, генетические алгоритмы, …

Модуль random

- randint – целое число из заданного диапазона, **включая** конец
- choice – случайный элемент из заданной коллекции
- shuffle – случайное перемешивание списка
- sample – случайные k различных элементов коллекции
- Различные распределения случайных вещественных чисел

Время и дата

Модули

- calendar
- datetime
- time

Строки: str

Строки: создание

```
s1 = 'string'
s2 = "That's a string \n too!"
s3 = str(25)
s4 = s1 + s2
s5 = ""                  # пустая строка
s6 = ":)" * 100          # 100 смайликов
length = len(s6)
s7 = '''длинная
строка'''
```

Элементы строк

```
>>> s = 'My string'
```

	s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]
s	M	y		s	t	r	i	n	g
	s[-9]	s[-8]	s[-7]	s[-6]	s[-5]	s[-4]	s[-3]	s[-2]	s[-1]

```
>>> s[0], s[3], s[-1], s[-9], s[-0])
    ('M', 's', 'g',   'M',   'M')
```

Срезы (slices)

- s[a:b] = s[a] + s[a+1] + s[a+2] + … + s[b-1]

  (начинаем с позиции *a*, заканчиваем **до** позиции *b*)

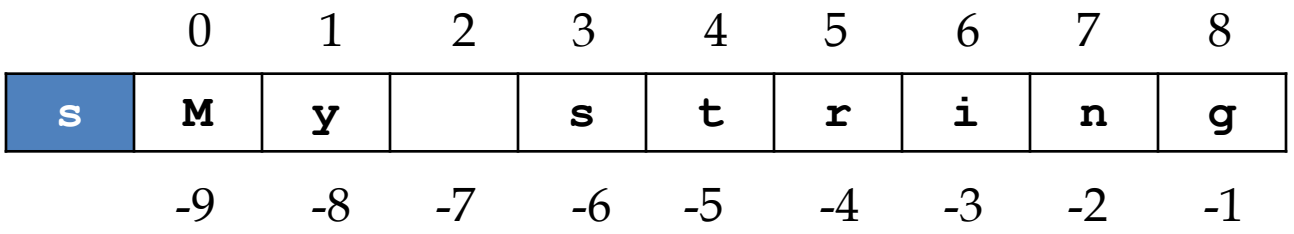
  s[3:7] = s[3] + s[4] + s[5] + s[6]

- s[a:b:c] = s[a] + s[a+c] + s[a+2c] + s[a+3c] + …

  (начинается с *a*, идем с шагом *c,* не доходя до позиции *b*)

  s[3:7:2] = s[3] + s[5]

- s[a:b] + s[b:c] = s[a:c]



Срез	Значение
s[3:8]	'strin'
s[5:]	‘ring'
s[:5] s[0:5]	'My st'
s[-8:-3]	'y str'
s[1:-3]	'y str'
s[:-1]	'My strin'
s[:]	'My string'
s[2:1]	''

Срез	Значение
s[3:8:2]	'srn'
s[3:100:2]	'srn'
s[3:8:1]	'strin'
s[8:3:-1]	‘gnirt'
s[8:3:-2]	‘git'
s[3:8:0]	Ошибка
s[::2]	'M tig'
s[::-1]	'gnirts yM'

Некоторые методы обработки строк

Синтаксис	Комментарий
s.find(sample, [start, [end]])	Возвращает индекс начала первого вхождения sample или -1 (rfind – последнее вхождение)
s.replace(from, to)	Заменяет все вхождения подстроки from на to
s.count(sample)	Возвращает количество (непересекающихся) вхождения sample
sep.join(список строк)	Склеивает все строки из списка в одну, разделяя их sep
s.split([chars])	Разрезает строку на части по любому из символов chars (без параметров – по пробельным символам, считая несколько пробельных символов подряд за один разделитель
s.strip([chars])	Удаляет все символы chars из начала и конца строки (lstrip, rstrip)

Строки – неизменяемые (immutable) объекты

НЕЛЬЗЯ!	МОЖНО!
s[len(s)] = 'a'	s = s + 'a'
s[0] = 'a'	s = 'a' + s[1:]
s[3] = 'a'	s = s[:3] + 'a' + s[4:]
удалить символ	s = s[:3] + s[4:]
вставить символ	s = s[:3] + 'a' + s[3:]
s.replace('a', 'b')	s = s.replace('a', 'b')
перевернуть строку	s = s[::-1]

Еще о строках

Метод **format** :

- форматированный вывод строк "похожий на printf"

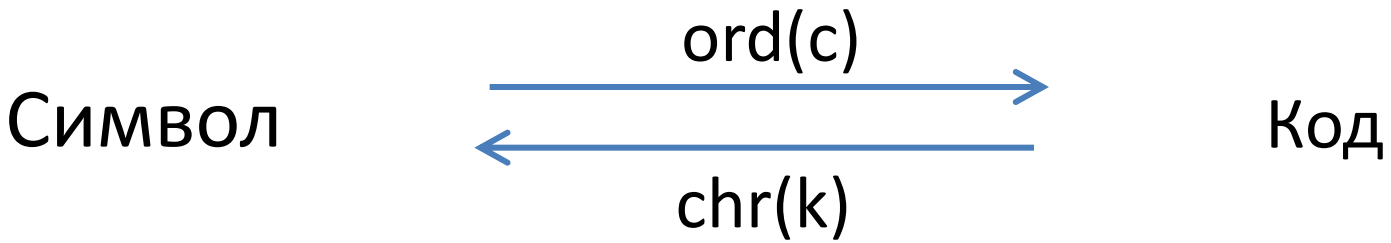
Модуль **re:**

- регулярные выражения

Встроенный тип данных **dict** (подробности позже):

```
color['apple'] = 'green'
```

Символы и коды



Контейнеры

Кортеж (tuple) – неизменяемый (immutable) массив

- Операции – как со строками: индексы, срезы, методы…
- Хранит любые объекты (на самом деле: ссылки на объекты), в том числе и изменяемые:
- a = (40,) #для одного элемента запятая обязательна!
- a = (1, 2, 'q', (3, 5), [4, 6], None)
- Доступ к элементам: a[1], a[3][1], a[4][0] (нумерация с нуля!)

Список (list) – изменяемый (mutable) массив

- s = [1, 2, 3, 'abc']
- s = list('abc') – список букв
- s = [] – пустой список
- s = [0] * 100 – список из ста нулей
- s = [[0] * 100 for i in range(100)]

Операции

- как с tuple или str

+ методы для изменения списка:

s.append(5), s.extend([6, 7, 8])

s.insert(i, x), del s[k], del s[i:j], s.remove(2), s.pop()

s[1] = 3, s[1:5] = [2, 8], s[5] = [],

s.sort() - упорядочивает сам список s по возрастанию

s.sort(reverse = True) - по убыванию

s.sort(key = f) - упорядочивает s, сравнивая между собой f(s[i])

s.reverse()

- s = [2*i for i in range(100)]
- s = input().split()
- s = list(map(int, input().split()))

Словарь (dict)

- D = {} или D = dict()
- D = {'spam': 2, 'eggs': 3}
- D = {'food': {'ham': 1, 'egg': 2}}
- D = dict(name='Bob', age=40)
- D = dict(zip(keyslist, valslist))
- D['eggs']
- D['food']['ham']
- 'eggs' in D
- D.keys()
- D.values()
- len(D) #Length: number of stored entries
- D[key] = 42 #Adding/changing keys

Множество (set)

- S = set()
- S = {5, 'abc', 100}
- S = set("apple")
- | & - ^
- S.add(x)
- 5 in S, 5 not in S
-

Функции

- Описание функции

```
def average(a, b, c):
    d = a + b + c
    return d/3

def average(s):
    return sum(s)/len(s)
```

- Использование функций

```
print(average(1,2,3))
print(average([1,2,3,4,5]))
t = average(s) + average(s[::2])
```

- Рекурсия

```
def product(s):
    if len(s) > 0:
        return product(s[:-1])*s[-1]
    else:
        return 1
```