Собственные заметки по Python

1. Lambda выражения

# рассмотри варианты использования lambda функций  
  
# классическое lambda выражение  
lambda\_func = lambda a, b: a \* b  
print(lambda\_func(2, 3))  
  
  
# в данном случае мы возвращаем сначала  
# lambda функцию с предустановленным параметром n  
# потом уже передаём значения в саму функцию  
def myfunc(n):  
 return lambda a: a \* n  
  
  
mydoubler = myfunc(2) # вернули lambda функцию  
print(mydoubler(11)) # передали значение в lambda функцию  
  
  
# изначально map возвращает просто iterable объект  
# который просто так не получится напечатать  
# lambda выражения можно использовать совместно с функциональным программирование  
temp\_first = list(map(lambda x: x \*\* 2, [1, 3, 5]))  
print(temp\_first)  
  
temp\_second = set(filter(lambda x: x > 5, [1, 3, 6, 7]))  
print(temp\_second)  
  
temp\_third = list(filter(lambda x: x > 9, map(lambda x: x \*\* 2, [1, 2, 3, 5, 6])))  
print(temp\_third)  
  
  
# сравнение lambda функции с обычной функцией  
# lambda функция имеет авто возврат  
# не имеет docstring  
# содержит только одну строку кода  
# у lambda непонятный возвращаемы тип  
temp = lambda x: x + 1  
print(temp(1), temp)  
  
def temp1(s): return s + 1  
print(temp1(4), temp1)  
  
  
# пример использования docstring  
# в lambda выражении нельзя использовать docstring  
def test\_doc():  
 *"""  
 it is doc string  
 :return:  
 """*print(test\_doc.\_\_doc\_\_)

1.1

# рассмотрим функциональное программирование  
# с применение lambda функций дальше  
  
from functools import reduce  
from itertools import zip\_longest  
from functools import reduce  
  
# если у нас два параметра .то передаём два списка для считывания данных  
test1 = list(map(lambda x, y: x + y, [1, 2, 3], [4, 5, 6]))  
print(test1)  
  
# аналогично можно добиться подобного результата с генератором списка  
test2 = [x + y for x, y in zip([1, 2, 3], [4, 5, 6])]  
print(test2)  
  
# функция zip(list1(), list2(), list3())  
# объединает в кортежи множество list  
# zip останавливается когда заканчивается самая коротка последовательность  
# zip\_longest() отработает до конца модуль itertools  
test3 = list(zip([1, 2, 6], [4, 5, 3], [6, 7, 9]))  
print(test3)  
  
test4 = zip\_longest([1, 2], ['a', 32.4], [1, 2, 3])  
print(list(test4))  
  
  
# reduce часть модуля functools  
# возвращает число  
# reduce возвращаемое значение может быть не только число  
test5 = reduce(lambda x, y: [y] + x, [1, 2, 3, 4], [])  
print(test5)

1.2

from functools import reduce  
  
# в функциии map и подобные можно передать  
# ссылку на другую функцию, так же надо передать точное число аргументов  
def temp1(a, b):  
 return a + b  
  
  
test1 = list(map(temp1, [1, 2, 3, 4], [1, 2, 3, 4]))  
print(test1)  
  
def temp2(a, b):  
 return a + b  
  
test2 = reduce(lambda x, y: y + x, [1, 2, 3], 5)  
print(test2)  
  
test3 = reduce(temp2, [1, 2, 3], 5)  
print(test3)  
  
def temp3(digit):  
 return 0  
  
test4 = list(filter(temp3, [1, 3, 45, 5]))  
print(test4)