



Лабораторные работы по курсу:

«Разработка Интернет Приложений»

Python. Функциональные возможности

Исполнитель:

Студент группы РТ5-51

Макаров А.В.

Преподаватель:

Гапанюк Ю.Е,

« _ » _____



Задача 1 (ex_1.py)

Необходимо реализовать генераторы `field` и `gen_random`

Генератор `field` последовательно выдает значения ключей словарей массива

Пример:

```
goods = [  
{ 'title': 'Ковер', 'price': 2000, 'color': 'green' },  
{ 'title': 'Диван для отдыха', 'color': 'black' }  
]
```

`field(goods, 'title')` должен выдавать 'Ковер', 'Диван для отдыха' `field(goods, 'title', 'price')`

должен выдавать `{ 'title': 'Ковер', 'price': 2000 }`, `{ 'title': 'Диван для отдыха' }`

1. В качестве первого аргумента генератор принимает `list`, дальше через `*args` генератор принимает неограниченное кол-во аргументов.
2. Если передан один аргумент, генератор последовательно выдает только значения полей, если поле равно `None`, то элемент пропускается
3. Если передано несколько аргументов, то последовательно выдаются словари, если поле равно `None`, то оно пропускается,

если все поля `None`, то пропускается целиком весь элемент Генератор `gen_random` последовательно выдает заданное количество случайных чисел в заданном диапазоне Пример:

`gen_random(1, 3, 5)` должен выдать 5 чисел от 1 до 3, т.е. примерно 2, 2, 3, 2, 1 В `ex_1.py` нужно вывести на экран то, что они выдают о дной строкой Генераторы должны располагаться в `librip/ gen.py`

```
[Running] python3 "/home/catmen/Документы/lab2_repo/lab4/ex_1.py"
```

```
Test:
```

```
Test:
```

```
{ 'title': 'Ковер' }  
{ 'title': 'Диван для отдыха' }  
{ 'title': 'Стелаж' }  
{ 'title': 'Вешалка для одежды' }
```

```
[Done] exited with code=0 in 0.108 seconds
```

```
[Running] python3 "/home/catmen/Документы/lab2_repo/lab4/ex_1.py"
```

```
Test:
```

```
Ковер
```

```
Диван для отдыха
```

```
Стелаж
```

```
Вешалка для одежды
```

```
Test:
```

```
{ 'color': 'green', 'title': 'Ковер' }  
{ 'color': 'black', 'title': 'Диван для отдыха' }  
{ 'color': 'white', 'title': 'Стелаж' }  
{ 'color': 'red' }  
{ 'color': 'green' }  
{ 'color': 'white', 'title': 'Вешалка для одежды' }
```

```
[Done] exited with code=0 in 0.13 seconds
```

Задача 2 (ex_2.py)

Необходимо реализовать итератор, который принимает на вход массив или генератор и итерируется по элементам, пропуская дубликаты. Конструктор итератора также принимает на вход именной `bool`-параметр

`ignore_case`, в зависимости от значения которого будут считаться одинаковыми строки в разном регистре. По умолчанию этот параметр равен `False`. Итератор не

должен модифицировать возвращаемые значения. Пример:

```
data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
```

`Unique(data)` будет последовательно возвращать только 1 и 2 МГТУ им. Н. Э. Баумана, кафедра `data = gen_random(1, 3, 10)`

`unique(gen_random(1, 3, 10))` будет последовательно возвращать только 1, 2 и 3 `data = ['a', 'A', 'b', 'B']`

`Unique(data)` будет последовательно возвращать только `a, A, b, B` `data = ['a', 'A', 'b', 'B']`

`Unique(data, ignore_case=True)` будет последовательно возвращать только `a, b` В `ex_2.py` нужно вывести на экран то, что они выдают о дной строкой. Важно продемонстрировать работу как

с массивами, так и с генераторами (`gen_random`). Итератор должен располагаться в `librip/iterators.py`

```
[Running] python3 "/home/catmen/Документы/lab2_repo/lab4/ex_2.py"
[1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
[2, 1, 2, 3, 1, 3, 1, 2, 2, 1]
[3, 3, 3, 3, 2, 3, 1, 1, 3, 3]
['b', 'a']
```

```
[Done] exited with code=0 in 0.213 seconds
```

Задача 3 (ex3.py)

Дан массив с положительными и отрицательными числами. Необходимо одной строкой вывести на экран массив,

отсортированный по модулю. Сортировку осуществлять с помощью функции *sorted* Пример:

`data = [4, -30, 100, -100, 123, 1, 0, -1, -4]`

Вывод: `[0, 1, -1, 4, -4, -30, 100, -100, 123]`

```
[Running] python3 "/home/catmen/Документы/lab2_repo/lab4/ex_3.py"
```

```
[0, 1, -1, 4, -4, -30, 100, -100, 123]
```

```
[Done] exited with code=0 in 0.167 seconds
```

Задача 4 (ex_4.py)

Необходимо реализовать декоратор `print_result` , который выводит на экран результат выполнения функции.

Файл `ex_4.py` не нужно изменять.

Декоратор должен принимать на вход функцию, вызывать её, печатать в консоль имя функции, печатать результат и возвращать значение.

Если функция вернула список (`list`), то значения должны выводиться в столбик.

Если функция вернула словарь (`dict`), то ключи и значения должны выводиться в столбик через

знак равно

Пример:

```
@print_result
```

```
def test_1():
```

```
    return 1
```

```
@print_result
```

```
def test_2():
```

```
    return 'iu'
```

```
@print_result
```

```
def test_3():
```

```
    return {'a': 1, 'b': 2}
```

```
@print_result
```

```
def test_4():
```

```
    return [1, 2]
```

```
test_1()
```

```
test_2()
```

```
test_3()
```

```
test_4()
```

На консоль выведется:

```
[Running] python3
"/home/catmen/Документы/lab2_repo/lab4/ex_4.py" test_1
1
test_2
iu
test_3
a = 1
b = 2
test_4
1
2
```

```
[Done] exited with code=0 in 0.182 seconds
```

Задача 5 (ex_5.py)

Необходимо написать контекстный менеджер, который считает время работы блока и выводит его на экран

Пример:

```
with timer():
```

```
    sleep(5.5)
```

После завершения блока должно вывестись в консоль примерно 5.5

```
[Running] python3 "/home/catmen/Документы/lab2_repo/lab4/ex_5.py"
```

```
Затрачено время: 5.506
```

```
[Done] exited with code=0 in 5.715 seconds
```

Задача 6 (ex_6.py)

Мы написали все инструменты для работы с данными. Применим их на реальном примере, который мог возникнуть в жизни. В репозитории находится файл `data_light.json`. Он содержит облегченный список

вакансий в России в формате `json` (ссылку на полную версию размером ~ 1 Гб. в формате `xml` можно найти в файле `README.md`).

Структура данных представляет собой массив словарей с множеством полей: название работы, место, уровень зарплаты и т.д.

В `ex_6.py` дано 4 функции. В конце каждая функция вызывается, принимая на вход результат работы предыдущей. За счет декоратора `@print_result` печатается результат, а контекстный менеджер `timer` выводит время работы цепочки функций.

Задача реализовать все 4 функции по заданию, ничего не изменяя в файле-шаблоне. Функции `f1-f3` должны быть реализованы в 1 строку, функция `f4` может состоять максимум из 3 строк. Что функции должны делать:

1. Функция `f1` должна вывести отсортированный список профессий без повторений (строки в разном регистре считать равными). Сортировка должна игнорировать регистр. Используйте наработки из предыдущих заданий.
2. Функция `f2` должна фильтровать входной массив и возвращать только те элементы, которые начинаются со слова “программист”. Иными словами нужно получить все специальности, связанные с программированием. Для фильтрации используйте функцию `filter`.
3. Функция `f3` должна модифицировать каждый элемент массива, добавив строку “с опытом Python” (все программисты должны быть знакомы с Python). Пример: Программист C# с опытом Python. Для модификации используйте функцию `map`.
4. Функция `f4` должна сгенерировать для каждой специальности зарплату от 100 000 до 200 000 рублей и присоединить её к названию специальности. Пример: Программист C# с опытом Python, зарплата 137287 руб. Используйте `zip` для обработки пары специальность — зарплата.

```
инженер-программист (орехово-зубовский филиал) с опытом Python, зарплата 159391 руб.  
инженер-программист 1 категории с опытом Python, зарплата 173331 руб.  
инженер-программист ккт с опытом Python, зарплата 108956 руб.  
инженер-программист плиз с опытом Python, зарплата 139886 руб.  
инженер-программист сапоу (java) с опытом Python, зарплата 139101 руб.  
инженер-электронщик (программист асу тп) с опытом Python, зарплата 175984 руб.  
педагог программист с опытом Python, зарплата 172200 руб.  
помощник веб-программиста с опытом Python, зарплата 123314 руб.  
программист с опытом Python, зарплата 145063 руб.  
программист / senior developer с опытом Python, зарплата 139010 руб.  
программист 1с с опытом Python, зарплата 110560 руб.  
программист с# с опытом Python, зарплата 169095 руб.  
программист с++ с опытом Python, зарплата 160475 руб.  
программист с++/с#/java с опытом Python, зарплата 183608 руб.
```

программист/ junior developer с опытом Python, зарплата 110396 руб.
программист/ технический специалист с опытом Python, зарплата 146606 руб.
программист-разработчик информационных систем с опытом Python, зарплата 191345 руб.
системный программист (с, linux) с опытом Python, зарплата 135678
руб. старший программист с опытом Python, зарплата 170868 руб.
Затрачено время: 7.774

[Done] exited with code=0 in 8.505 seconds

Листинг

/lab4/

/ex1.py

```
from librip.gens import field
from librip.gens import gen_random
goods = [
    {'title': 'Ковер', 'price': 2000, 'color': 'green'},
    {'title': 'Диван для отдыха', 'price': 5300, 'color': 'black'},
]
```

```
test = field(goods, 'title')
print("field1:")
for string in test:
    print(string)
```

```
test = field(goods, 'title', 'price')
print("field2:")
for string in test:
    print(string)
```

```
print("\nGenRandom:")
print(gen_random(1, 3, 5))
```

/ex2.py

#!/usr/bin/env python3

```
from librip.gens import gen_random
from librip.gens import gen_random_one_string
from librip.iterators import unique
```

```
data1 = list(i for i in unique([1, 1, 1, 1, 1, 2, 2, 2, 2, 2], ignore_case=True))
data2 = list(i for i in unique(gen_random(1, 3, 10), ignore_case=True))
```

```
data4 = list(i for i in unique(['a', 'A', 'b', 'B'], ignore_case=True))
data5 = list(i for i in unique(['a', 'A', 'b', 'B'], ignore_case=False))
print(data1)
print(data2)
print(data4)
print(data5)
```

#Реализация задания 2

/ex3.py

#!/usr/bin/env python3

```
data = [4, -30, 100, -100, 123, 1, 0, -1, -4]
```

```
print(sorted(data, key = abs))
```

Реализация задания 3

```
/ex_4.py
from librip.decorators import print_result
```

```
@print_result
def test_1():
    return 1
```

```
@print_result
def test_2():
    return 'iu'
```

```
@print_result
def test_3():
    return {'a': 1, 'b': 2}
```

```
@print_result
def test_4():
    return [1, 2]
```

```
test_1()
test_2()
test_3()
test_4()
```

```
/ex_5.py
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
from time import sleep
from librip.ctxmgrs import timer
```

```
with timer():
    sleep(5.5)
```

```
/ex_6.py
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

```
import json
import sys
import argparse
from librip.ctxmgrs import timer
from librip.decorators import print_result
from librip.gens import field, gen_random
from librip.iterators import unique
```

```
def create_Parser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-P', '--path', default = 'data_light.json') #путь к источнику
    return parser
```

```
#проверка аргументов командной строки
```

```
if __name__ == "__main__":
    if len(sys.argv) < 1:
        print('Exception:: Недостаточно параметров командной строки')
        raise SystemExit(1)
    parser = create_Parser()
    namespace = parser.parse_args(sys.argv[1:])
```

```
path = namespace.path
```

```
with open(path) as f:
    data = json.load(f)
```

```
@print_result
def fl(arg):
    return sorted(unique([i for i in field(arg, 'job-name')], ignore_case = False), key=lambda
x:x.lower())
```

```

@print_result
def f2(arg):
    return filter(lambda x: "программист" in x, arg)

@print_result
def f3(arg):
    return list(map(lambda x: "{0} с опытом Python".format(x), arg))

@print_result
def f4(arg):
    return [{"0}, зарплата {1} руб.".format(x, y) for x, y in zip(arg, list(gen_random(100000,
200000, len(arg))))]

with timer():
    f4(f3(f2(f1(data))))

/librip/
    /ctxmng.rs.py

# -*- coding: utf-8 -*-

import time

class timer:
    start = 0
    def __enter__(self):
        self._start = time.time()
    def __exit__(self, exp_type, exp_value, traceback):
        print("время:", round(time.time() - self._start, 3))

    /decorators.py
def print_result(func):
    def decarted(*args, **kwargs):
        print(func.__name__)
        result = func(*args, **kwargs)
        if type(result) == list:
            for value in result:
                print(value)
        elif type(result) == dict:
            for parameter, value in result.items():
                print(parameter, "=", value)
        else:
            print(result)
        return result
    return decarted

    /gens.py

import random

def field(items, *args):
    assert len(args) > 0
    if len(args) == 1:
        for value in items:
            temp = value.get(args[0])
            if temp is not None:
                yield temp
    else:
        for value in items:
            temp = {key: value.get(key) for key in args if value.get(key) is not None}
            if len(temp) != 0:
                yield temp
    # Необходимо реализовать генератор

def gen_random_one_string(begin, end, num_count):
    return list(random.randint(begin, end) for n in range(num_count))

def gen_random(begin, end, num_count):
    mass = list()
    while num_count != 0:
        mass.append(random.randint(begin, end))

```

```
    num_count -= 1
    return mass
```

/iterators.py

```
# -*- coding: utf-8 -*-
```

```
# Итератор для удаления дубликатов
```

```
class unique(object):
    def __init__(self, items, **kwargs):
        self.iter = len(items)
        self.items = items
        self.ignore_case = kwargs.get('ignore_case')

    def __next__(self):
        if self.ignore_case:
            while len(self.items) > 0:
                item = self.items.pop()
                try:
                    self.items.index(item)
                except ValueError:
                    return item
            else:
                while len(self.items) > 0:
                    flag = self.ignore_case
                    item = self.items.pop().lower()
                    for temp in self.items:
                        if item == temp.lower():
                            flag = False
                            break
                    if not flag:
                        return item
                raise StopIteration
        else:
            while len(self.items) > 0:
                item = self.items.pop()
                try:
                    self.items.index(item)
                except ValueError:
                    return item
            else:
                raise StopIteration

    # Нужно реализовать next

    def __iter__(self):
        return self
    #
```