# Лабораторная работа 6

# Выравнивание списка, состоящего из итерируемых объектов, на основе рекурсии

Компьютерная математика II, ММФ, БГУ

Лаврова О.А.; идея Козак А.В., Кушнеров А.В., апрель 2024

## Навыки

**Python**: итерируемые объекты; метод `__iter__` ; функция `isinstance` ; функция `hasattr` ; обработка исключений: операторы `raise` , `try` / `except` ; встроенное исключение `ValueError` ; рекурсивная функция

# Определения

**Рекурсивная функция** -- это функция, которая вызывает сама себя, и при каждом очередном вызове может использовать объекты, созданные во время предыдущего вызова.

**Стек вызовов** -- это область памяти, в которой выполняются функции. При каждом вызове функции создается **фрейм** -- фрагмент памяти, в котором содержится:

- информация о текущем выполнении функции;
- объекты, полученные функцией для обработки;
- локальные пременные;
- сведения о строке программы, к которой нужно вернуться после выполнения функции.

Фреймы добаляются и удаляются по принципу стека (last in, first out).

Рекурсивная функция, которая бесконечно вызывает себя, переполняет стек вызовов (stack overflow). Интерпретатор Python отслеживает переполнение стека

In [1]:
```
def recursive():
    recursive()
recursive()
```

```
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
Cell In[1], line 3
      1 def recursive():
      2     recursive()
----> 3 recursive()

Cell In[1], line 2, in recursive()
      1 def recursive():
----> 2     recursive()

Cell In[1], line 2, in recursive()
      1 def recursive():
----> 2     recursive()

    [... skipping similar frames: recursive at line 2 (2970 times)]

Cell In[1], line 2, in recursive()
      1 def recursive():
----> 2     recursive()

RecursionError: maximum recursion depth exceeded
```

Функции `getrecursionlimit` и `setrecursionlimit` из модуля `sys` позволяют получить глубину рекурсии и изменить ее

In [2]:
```
import sys
sys.getrecursionlimit()
```

Out[2]:
```
3000
```

Для того, чтобы рекурсивная функция реализовывала конечное число вызовов, необходимо в рекурсивной функции обрабатывать ситуацию, когда функция возвращает результат. Такая "граничная" ситуация называется базой рекурсии.

In [3]:
```
def sum(n):
    """суммирует числа от 0 до n>=0"""
    match n:
        case 0: return n # база рекурсии
        case _: return n + sum(n-1)
sum(0), sum(2), sum(10)
```

Out[3]:
```
(0, 3, 55)
```

Альтернатиное определение рекурсии с использованием синтаксиса lambda-функций

In [4]:
```
sum = lambda n: n + sum(n-1) if n else n
sum(0), sum(2), sum(10)
```

Out[4]:
```
(0, 3, 55)
```

# Постановка задачи

\color{red}\text{Напишите} пользовательскую функцию `flatten_it(nested_obj, gen)` выравнивания списка *на основе рекурсии*, которая для списка `nested_obj`, содержащего различные итерируемые объекты (список, кортеж, строка, словарь, генераторный объект), возвращает либо список из всех элементов итерируемых объектов без сохранения структуры итерируемых объектов, если `gen=False`, либо функция возвращает генераторный объект, если `gen=True`, который генерирует по запросу элементы итерируемых объектов без сохранения структуры итерируемых объектов.

Приведем различные примеры вызовов функции `flatten_it`.

*Пример 1*. Если вложенным итерируемым объектом является **список** или **кортеж**, то функция `flatten_it` возвращает список, состоящий из элементов списка или кортежа без сохранения структуры вложенности для элементов. Например,

```
flatten_it([1,(2,[3]),[[4]]], gen=False)
```

возвращает

```
[1,2,3,4]
```

*Пример 2*. Если вложенным итерируемым объектом является **строка**, то функция `flatten_it` оставляет строку без изменений, не итерируя ее по символам. Например,

```
flatten_it(["abc",["de"]], gen=False)
```

возвращает

```
["abc","de"]
```

*Пример 3*. Если вложенным итерируемым объектом является **генераторный объект**, то функция `flatten_it` сохраняет в результирующем списке сгенерированные элементы. Например,

```
flatten_it([[[range(5)]]], gen=False)
```

возвращает

```
[0,1,2,3,4]
```

*Пример 4*. Если вложенным итерируемым объектом является **словарь**, то функция `flatten_it` сохраняет ключи и соответствующие им объекты в качестве элементов результирующего списка. Например,

```
flatten_it([{1: {2,3}, 4: [5]}], gen=False)
```

возвращает

```
[1,2,3,4,5]
```

# Задание 6.1. Выравнивание вложенных списков

Предположим, что нам нужно выровнять вложенный список

In [1]:
```
it_list = [[1],[2,[3]],[[[[[4]]]]],5,6,7]
```

Напишем *рекурсивную функцию* `flatten_list_v1(nested_obj)`, которая выравнивает вложенный список `nested_obj` и возвращает список, содержащий только элементы исходного списка без сохранения его структуры.

Некоторые аспекты реализации:

- для проверки, является ли текущий объект `item` списком или нет, будем использовать функцию `isinstance(item, list)`;
- для формирования результирующего списка `result` будем использовать дополненное присваивание `+=` для списков на основе операции конкатенации `+`.

In [2]:
```
?isinstance
```

```
Signature: isinstance(obj, class_or_tuple, /)
Docstring:
Return whether an object is an instance of a class or of a subclass thereof.

A tuple, as in ``isinstance(x, (A, B, ...))``, may be given as the target to
check against. This is equivalent to ``isinstance(x, A) or isinstance(x, B)
or ...`` etc.
Type:      builtin_function_or_method
```

In [3]:
```
def flatten_list_v1(nested_obj):
    result = []
    if isinstance(nested_obj,list):
        for item in nested_obj:
            result += flatten_list_v1(item) # рекурсия
    else:
        result += [nested_obj]
    return result
```

Протестируем функцию `flatten_list_v1`

In [4]:
```
flatten_list_v1(it_list)
```

Out[4]:
```
[1, 2, 3, 4, 5, 6, 7]
```

### Задание 6.1.1. Генераторная функция flatten_list_v2

На основе функции `flatten_list_v1` \color{red}\text{напишите} генераторную функцию `flatten_list_v2(nested_obj)`, которая выравнивает вложенный список и возвращает генератнорый объект для элементов вложенного списка.

Реализация генераторной функции `flatten_list_v2` будет проще реализации функции `flatten_list_v1`, так как не нужно объединять все объекты в единый результирующий список.

Протестируйте функцию `flatten_list_v2`

In [6]:
```
[x for x in flatten_list_v2(it_list)]
```

Out[6]:
```
[1, 2, 3, 4, 5, 6, 7]
```

### Задание 6.1.2. Функция flatten_list

\color{red}\text{Объедините} две пользовательские функции `flatten_list_v1` и `flatten_list_v2` в одну пользовательскую функцию `flatten_list(nested_obj, gen)`, которая возвращает либо список из элементов вложенных списков без сохранения структуры, если `gen=False`, либо возвращает генераторный объект, если `gen=True`. Стандартным значением для аргумента `gen` является объект `False`.

### Задание 6.1.3. Документирование функции flatten_list

\color{red}\text{Напишите} строки документации для функции `flatten_list`. Напишите комментарии для каждой строки кода функции `flatten_list`.

### Задание 6.1.4. Тестирование функции flatten_list

Приведите три примера вызова функции `flatten_list` для различных значений аргументов в предположении, что корректность вводимых данных гарантируется.

# Задание 6.2. Выравнивание вложенных итерируемых объектов произвольного типа

Создадим различные итерируемые объекты: список, кортеж, строка, словарь, генераторный объект, файловый объект

In [7]:
```
it_list
```

Out[7]:
```
[[1], [2, [3]], [[[[[4]]]]], 5, 6, 7]
```

In [8]:
```
it_tuple = (((7),(8)),(9),10)
```

In [9]:
```
it_str = "abcdefgh"
```

In [10]:
```
it_dict = {"key1": 11, "key2": {"key3": 12}}
```

In [11]:
```
it_gen = ([i,i**2,i**3] for i in range(5))
```

In [12]:
```
it_file = open("flatten_file.txt")
```

Объединим все итерируемые объекты в список

In [13]:
```
it = [it_list, it_tuple, it_str, it_dict, it_gen, it_file]
```

Все итерируемые объекты имеют метод `__iter__` для создания объекта итератора

In [14]:
```
['__iter__' in dir(x) for x in it]
```

Out[14]:
```
[True, True, True, True, True, True]
```

In [15]:
```
[hasattr(x, '__iter__') for x in it]
```

Out[15]:
```
[True, True, True, True, True, True]
```

In [16]:
```
?hasattr
```

```
Signature: hasattr(obj, name, /)
Docstring:
Return whether the object has an attribute with the given name.

This is done by calling getattr(obj, name) and catching AttributeError.
Type:      builtin_function_or_method
```

## Задание 6.2.1. Функция flatten_it_v1

На основе функции `flatten_list_v1(nested_obj)` \color{red}\text{напишите} пользовательскую функцию `flatten_it_v1(nested_obj)` одного аргумента, которая выравнивает вложенную структуру из различных итерируемых объектов и возвращает список, содержащий только элементы без сохранения структуры.

Для этого вместо проверки `isinstance(nested_obj, list)` будем использовать проверку на итерируемость объекта `hasattr(nested_obj, '__iter__')` .

При этом дополнительно учтем, что если итерируемым объектом является

- строка, то ее мы не итерируем;
- словарь, то для его итерирования нужно выполнить дополнительные преобразования.

In [17]:
```
def flatten_it_v1(nested_obj):
    result = []
    if hasattr(nested_obj, '__iter__'):
        ...
    else:
        result += [nested_obj]
    return result
```

Протестируйте функцию `flatten_it_v1`

In [18]:
```
print(flatten_it_v1(it))
```

```
[]
```

### Задание 6.2.2. Генераторная функция flatten_it_v2

На основе функций `flatten_it_v1` и `flatten_list_v2` \color{red}\text{напишите} пользовательскую функцию `flatten_it_v2(nested_obj)`, которая выравнивает вложенную структуру из различных итерируемых объектов и возвращает генератнорый объект элементов без сохранения структуры.

### Задание 6.2.3. Функция flatten_it

\color{red}\text{Объедините} две пользовательские функции `flatten_it_v1` и `flatten_it_v2` в одну функцию `flatten_it(nested_obj, gen)`, которая возвращает либо список из элементов вложенных объектов без сохранения структуры, если `gen=False`, либо возвращает генераторный объект, если `gen=True`. Стандартным значением для аргумента `gen` является объект `False`.

### Задание 6.2.4. Документирование flatten_it

\color{red}\text{Напишите} строки документации для функции `flatten_it`. ** \color{red}\text{Напишите} комментарии для каждой строки кода функции `flatten_it`.

### Задание 6.2.5. Тестирование flatten_it

Приведите три примера вызова функции `flatten_it` для различных значений аргументов в предположении, что корректность вводимых данных гарантируется.

# Задание 6.3. Обработка циклических объектов

Циклическими объектами в Python могут быть словари и списки. Циклический список содержит элемент, который ссылается на весь список. Циклический словарь содержит элемент <ключ>:<значение>, <значение> которого сылается на весь словарь.

Создадим циклический объект списка

In [19]:
```
it_list_cyclic = it_list[:]
it_list_cyclic.append(it_list_cyclic)
```

Для отображения циклических объектов в Python используется специальный синтаксис

In [20]:
```
it_list_cyclic
```

Out[20]:
```
[[1], [2, [3]], [[[[[4]]]]], 5, 6, 7, [...]]
```

Обработаем дополнительно ситуацию, когда вложенный список является циклическим объектом.

Для этого изменим код пользовательской функции `flatten_list_v1` из Задания 6.1, создав исключение `ValueError` с помощью оператора `raise` , когда текущий элемент `item` списка `nested_obj` ссылается на этот же список

In [21]:
```
def flatten_list_v1(nested_obj):
    result = []
    if isinstance(nested_obj,list):
        for item in nested_obj:
            if item == nested_obj:
                raise ValueError("Cyclic list is found")  # генерация исключения
            else:
                result += flatten_list_v1(item)
    else:
        result += [nested_obj]
    return result
```

Протестируем функцию `flatten_list_v1`

In [22]:
```
flatten_list_v1(it_list)
```

Out[22]:
```
[1, 2, 3, 4, 5, 6, 7]
```

In [23]:
```
flatten_list_v1(it_list_cyclic)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[23], line 1
----> 1 flatten_list_v1(it_list_cyclic)

Cell In[21], line 6, in flatten_list_v1(nested_obj)
      4 for item in nested_obj:
      5     if item == nested_obj:
----> 6         raise ValueError("Cyclic list is found")  # генерация исключения
      7     else:
      8         result += flatten_list_v1(item)

ValueError: Cyclic list is found
```

Для того, чтобы стандартное сообщение об ошибке не выводилось, *перехватим* созданное нами исключение `ValueError` внутри функции `flatten_list_v1` с использованием оператора `try` в сочетании с `except`. Если внутри блока `try` возникнет исключение `ValueError`, то процесс выполнения кода перейдет в блок оператора `except`, для которого указано исключение `ValueError`

In [24]:
```
try:
    print(flatten_list_v1(it_list_cyclic))
except ValueError as e:
    print(e)
```

```
Cyclic list is found
```

## Задание 6.3.1. Доопределение функции flatten_it

\color{red}\text{Доопределите} пользовательскую функцию `flatten_it`, добавив генерацию исключения `ValueError` с помощью оператора `raise` при обработке циклических списков и циклических словарей.

## Задание 6.3.2. Тестирование функции flatten_it

\color{red}\text{Протестируйте} функцию `flatten_it`, перехватывая исключение `ValueError` с помощью операторов `try` / `except` для циклических объектов различных типов.