Знакомство с языком Python (семинары)

Задание 1. Поиск элемента

Пользователь вводит искомый ключ. Если он хочет, то может ввести максимальную глубину — уровень, до которого будет просматриваться структура.

Напишите функцию, которая находит заданный пользователем ключ в словаре и выдаёт значение этого ключа на экран. По умолчанию уровень не задан. В качестве примера можно использовать такой словарь:

```
site = {
'html': {
'head': {
'title': 'Мой сайт'
},
'body': {
'h2': 'Здесь будет мой заголовок',
'div': 'Тут, наверное, какой-то блок',
'p': 'А вот здесь новый абзац'
}
}
}
```

Пример 1

Введите искомый ключ: head

Хотите ввести максимальную глубину? Y/N: n

Значение ключа: {'title': 'Мой сайт'}

Пример 2

Введите искомый ключ: head

Хотите ввести максимальную глубину? Y/N: y

Введите максимальную глубину: 1

Значение ключа: None

Подсказка № 1

Создайте функцию `find_key`, которая будет принимать четыре параметра: `struct` — словарь, в котором нужно искать ключ, `key` — искомый ключ, `max_depth` — максимальная глубина поиска (по умолчанию `None`), и `depth` — текущая глубина поиска (по умолчанию 1).

Подсказка № 2

Начните реализацию функции с проверки, превышает ли текущая глубина поиска значение `max_depth`. Если превышает, верните `None`, так как дальше искать не нужно.

Подсказка № 3

Проверьте, находится ли искомый ключ в текущем словаре (`struct`). Если да, верните соответствующее значение. Если нет, продолжайте поиск по вложенным структурам.

Подсказка № 4

Для поиска во вложенных структурах используйте рекурсию. Пройдитесь по всем значениям словаря и, если значение является словарём, вызовите функцию `find_key` рекурсивно, увеличив текущую глубину на 1.

Эталонное решение:

```
# Функция для поиска ключа в словаре с учётом глубины поиска

def find_key(struct, key, max_depth=None, depth=1):

    result = None  # Переменная для хранения найденного значения



    # Если указана максимальная глубина и текущая глубина превышает
её, прекращаем поиск

    if max_depth and max_depth < depth:

        return result



    # Если ключ найден в текущем уровне словаря, возвращаем его
значение

    if key in struct:

        return struct[key]
```

```
    # Рекурсивный поиск по вложенным словарям
    for sub_struct in struct.values():
        if isinstance(sub_struct, dict):
            result = find_key(sub_struct, key, max_depth,
depth=depth + 1)
            if result:
                break

    return result


# Пример словаря с вложенными структурами
site = {
    'html': {
        'head': {
            'title': 'Мой сайт'
        },
        'body': {
            'h2': 'Здесь будет мой заголовок',
            'div': 'Тут, наверное, какой-то блок',
            'p': 'А вот здесь новый абзац'
        }
    }
}


while True:
    key = input('Введите искомый ключ: ')  # Запрашиваем ключ у
пользователя
```

```
    answer = input('Хотите ввести максимальную глубину? Y/N: ')  # 
Проверяем, хочет ли пользователь ограничить глубину поиска

    if answer.lower() == 'y':

        max_depth = int(input('Введите максимальную глубину: '))  # 
Если да, запрашиваем максимальную глубину

    else:

        max_depth = None  # Если нет, устанавливаем максимальную 
глубину как None



    # Выводим найденное значение ключа или None, если ключ не найден

    print('Значение ключа:', find_key(struct=site, 
max_depth=max_depth, key=key))
```

Задача 2. Глубокое копирование

Вы сделали для заказчика структуру сайта по продаже телефонов:

```
site = {
'html': {
'head': {
'title': 'Куплю/продам телефон недорого'
},
'body': {
'h2': 'У нас самая низкая цена на iPhone',
'div': 'Купить',
'p': ‘Продать'
}
}
}
```

Заказчик рассказал своим коллегам на рынке, и они захотели такой же сайт для своих товаров. Вы посчитали, что это лёгкая задача, и быстро принялись за работу.

Напишите программу, которая запрашивает у клиента количество сайтов, затем названия продуктов, а после каждого запроса выводит на экран активные сайты.

Условия:

- учтите, что функция должна уметь работать с разными сайтами (иначе вам придётся переделывать программу под каждого заказчика заново);
- вы должны получить список, хранящий сайты для разных продуктов (а значит, для каждого продукта нужно будет первым делом выполнить **глубокое** копирование сайта).

Пример вывода

Сколько сайтов: 2

Введите название продукта для нового сайта: iPhone

Сайт для iPhone:

```
site = {
'html': {
'head': {
'title': 'Куплю/продам iPhone недорого'
},
'body': {
'h2': 'У нас самая низкая цена на iPhone',
'div': 'Купить',
'p': ‘Продать'
}
}
```

```
}
```

Введите название продукта для нового сайта: Samsung

Сайт для iPhone:

```
site = {
'html': {
'head': {
'title': 'Куплю/продам iPhone недорого'
},
'body': {
'h2': 'У нас самая низкая цена на iPhone',
'div': 'Купить',
'p': ‘Продать'
}
}
}
```

Сайт для Samsung:

```
site = {
'html': {
'head': {
'title': 'Куплю/продам Samsung недорого'
},
'body': {
'h2': 'У нас самая низкая цена на Samsung',
```

'div': 'Купить',

'p': ‘Продать'

}

}

}

Обратите внимание, что на первой итерации выводится только один сайт (для iPhone), а на второй итерации — оба сайта (и для iPhone и для Samsung). Чтобы это реализовать, нужно сохранять сайты в списке и каждый раз печатать все его элементы.

Подсказка № 1

Перед началом создания программы определите количество необходимых сайтов. Для этого используйте функцию `input` и сохраните результат в переменной `sites_count`

Подсказка № 2

Создайте функцию `make_site`, которая принимает название продукта и создает структуру сайта, заменяя в ней соответствующие значения. Для этого используйте глубокое копирование (`copy.deepcopy`) для копирования исходной структуры сайта.

Подсказка № 3

В функции `make_site` замените значения ключей 'title' и 'h2' на соответствующие строки с названием продукта. Для этого используйте функцию `change_value`, которая рекурсивно ищет и заменяет значение ключа в словаре.

Подсказка № 4

Используйте список `sites` для хранения созданных сайтов. После создания каждого сайта добавляйте его в список и выводите все сайты из списка с помощью функции `display_struct`, которая рекурсивно выводит все ключи и значения словаря.

Подсказка № 5

Для корректного вывода структур в `display_struct` используйте рекурсию и параметр `spaces`, чтобы определять отступы. Если значение словаря является словарем, то сделайте рекурсию с передачей части значения структуры словаря.

Эталонное решение:

```
import copy
```

```
# Исходная структура сайта
site = {
    'html': {
        'head': {
            'title': 'Куплю/продам телефон недорого'
        },
        'body': {
            'h2': 'У нас самая низкая цена на iPhone',
            'div': 'Купить',
            'p': 'Продать'
        }
    }
}

# Функция для замены значения в структуре словаря
def change_value(struct, key, value):
    if key in struct:
        struct[key] = value
    else:
        for sub_struct in struct.values():
            if isinstance(sub_struct, dict):
                change_value(sub_struct, key, value)

    return struct

# Функция для отображения структуры сайта
```

```
def display_struct(struct, spaces=1):

    for key, value in struct.items():

        if isinstance(value, dict):

            print(' ' * spaces, key)

            display_struct(value, spaces + 3)

        else:

            print("{}{} : {}".format(' ' * spaces, key, value))



# Функция для создания сайта под конкретный продукт

def make_site(name):

    struct_site = copy.deepcopy(site)  # Глубокое копирование
исходного сайта

    new_title = 'Куплю/продам {} недорого'.format(name)  # Изменяем
заголовок

    struct_site = change_value(struct_site, 'title', new_title)

    new_h2 = 'У нас самая низкая цена на {}'.format(name)  #
Изменяем заголовок второго уровня

    struct_site = change_value(struct_site, 'h2', new_h2)



    return struct_site



# Основная часть программы

sites = []

sites_count = int(input('Сколько сайтов: '))

for _ in range(sites_count):

    product_name = input('Введите название продукта для нового
сайта: ')

    new_site = make_site(product_name)
```

```
    sites.append(new_site)

    for i_site in sites:

        display_struct(i_site)
```

Задача 3. Продвинутая функция sum

Как вы знаете, в Python есть полезная функция sum, которая умеет находить сумму элементов списков. Иногда базовых возможностей функций не хватает для работы и приходится их усовершенствовать.

Напишите свою функцию sum, которая должна быть более гибкой, чем стандартная. Она должна уметь складывать числа:

- из списка списков,
- набора параметров.

Основной код оставьте пустым или закомментированным (используйте его только для тестирования).

Примеры вызовов функции

`sum([[1, 2, [3]], [1], 3])`

`Ответ в консоли: 10`

`sum(1, 2, 3, 4, 5)`

`Ответ в консоли: 15`

Подсказка № 1

Начните с создания функции `my_sum`, которая принимает произвольное количество аргументов. Для этого используйте `*args`, чтобы функция могла принимать как одиночные числа, так и вложенные структуры данных

Подсказка № 2

Для суммирования чисел и рекурсивного вызова функции при обнаружении списков или кортежей создайте переменную `total_sum`, которая будет хранить итоговую сумму. Начальное значение переменной — 0.

Подсказка № 3

Используйте цикл для перебора элементов в `args`. Если элемент является числом, добавьте его к `total_sum`. Проверить это можно с помощью функции `isinstance(i_elem, int)`.

Подсказка № 4

Если элемент является списком или кортежем, используйте рекурсивный вызов функции `my_sum` для суммирования всех вложенных чисел. Результат вызова добавьте к `total_sum`.

Эталонное решение:

```
def my_sum(*args):
    total_sum = 0  # Инициализация переменной для хранения суммы

    for i_elem in args:
        # Проверка, является ли элемент целым числом
        if isinstance(i_elem, int):
            total_sum += i_elem  # Добавление числа к общей сумме
        # Проверка, является ли элемент списком или кортежем
        elif isinstance(i_elem, (list, tuple)):
            # Рекурсивный вызов функции для суммирования элементов внутри списка или кортежа
            for x in i_elem:
                total_sum += my_sum(x)

    return total_sum

# Основной код для тестирования
#print(my_sum([[1, 2, [3]], [1], 3]))  # Ожидаемый результат: 10
#print(my_sum(1, 2, 3, 4, 5))  # Ожидаемый результат: 15
```

Задача 4. Список списков

Дан список:

`nice_list = [1, 2, [3, 4], [[5, 6, 7], [8, 9, 10]], [[11, 12, 13], [14, 15], [16, 17, 18]]]`

Напишите рекурсивную функцию, которая раскрывает все вложенные списки, то есть оставляет только внешний список.

Ответ: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]`

Функция должна получать список и возвращать его раскрытую версию (не нужно добавлять элементы в список, записанный в глобальную переменную, созданную снаружи функции).

Подсказка № 1

Начните с создания функции `flatten`, которая принимает один аргумент — список с возможными вложенными списками. В функции создайте пустой список `result`, который будет хранить элементы без вложений.

Подсказка № 2

Используйте цикл `for` для перебора всех элементов в переданном списке `a_list`. Проверяйте каждый элемент, чтобы определить, является ли он целым числом или вложенным списком.

Подсказка № 3

Если элемент является числом (проверка с `isinstance(e, int)`), добавьте его в `result`. Если элемент — список, выполните рекурсивный вызов функции `flatten` с этим списком в качестве аргумента.

Подсказка № 4

Для добавления результатов рекурсивного вызова используйте метод `extend`, чтобы все элементы вложенного списка были добавлены в `result` как отдельные элементы.

Эталонное решение:

```
def flatten(a_list):

    # Инициализация пустого списка для хранения результата

    result = []



    # Перебор каждого элемента в списке
```

```
    for e in a_list:
        # Проверка, является ли элемент целым числом
        if isinstance(e, int):
            result.append(e)  # Добавление числа в результат
        else:
            # Рекурсивный вызов функции для вложенного списка
            result.extend(flatten(e))  # Раскрытие вложенных списков

    # Возвращаем окончательный результат
    return result

# Исходный список с вложенными элементами
nice_list = [1, 2, [3, 4], [[5, 6, 7], [8, 9, 10]], [[11, 12, 13], [14, 15], [16, 17, 18]]]

# Применение функции для получения списка с раскрытыми элементами
flattened_list = flatten(nice_list)

# Вывод результата
print(flattened_list)  # Ожидаемый результат: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
```