Лабораторная работа №2. Структуры данных и функции

Начало

Для решения заданий в лабушке, вам понадобится немного теории. Вы можете взять ее из моих конспектов или воспользоваться ссылками:

- https://pythonworld.ru/tipy-dannyx-v-python/vse-o-funkciyax-i-ix-argumentax.html
- https://python-scripts.com/functions-python
- https://devpractice.ru/python-lesson-7-work-with-list/

Непосредственно лабушка

Лабораторные работы выполняются в соответствии с вариантом. Вариант - это ваш порядковый номер в списке группы. Если вы в нем сомневаетесь, вы знаете, у кого его уточнить.

Лабораторные состоят из небольшого количества текста и задач. Решение задач хорошо скажется на вашем дальнейшем процессе обучения. Поэтому не советую на них забивать. Решения задач найти достаточно тяжело (не невозможно), поэтому старайтесь делать все сами.

Списки, словари, кортежи

Списки

Список представляет собой упорядоченную последовательность элементов, пронумерованных от 0. Список можно задать перечислением элементов списка в квадратных скобках.

Списки поддерживают разрезание и индексирование, могут быть вложенными, а также иметь множество полезных методов, которые можно вызывать из них.

```
my_list = [1, 2, 3]
another_list = [4, 5, 6]

print(my_list + another_list)
# [1, 2, 3, 4, 5, 6]
```

Списки также можно складывать.

Значения в списке могут быть любыми. Вы просто складываете два списка в один - и у вас получается новый объект-список.

Умножать списки также можно. Мало применимо, но можно. Но **обратите внимание** — деление и вычитание со списками не работают. При попытке сделать это вы получите ошибку.

Но почему? Дело в том, что объединение списков через **+** и повторение списков с помощью * — это очевидные действия, у которых есть аналоги в обычной жизни. Например, если я дам вам два списка покупок и попрошу их объединить, вы можете переписать все пункты этих списков на новый лист бумаги. Также, если я попрошу умножить этот список на 3, вы трижды перепишете на новый лист все пункты списка.

А если вы попытаетесь разделить список, скажем, из 6 элементов на две части, у вас появляется как минимум **15** вариантов, как это можно сделать. Деление не очевидно, поэтому делить списки нельзя. Равно как и складывать список с **НЕ**списочным значением. Проще говоря: к списку слово или цифру прибавить нельзя.

В списках вы можете переназначить элементы.

```
my_list = [1, 2, 3]
another_list = [4, 5, 6]

my_list[2] = 'три'
```

В данном случае, мы изменили элемент с индексом 2 на строчное значение.

Методы

Метод просто расширяет возможности работы со списками. У списков же есть несколько методов, которые мы будем часто использовать и первый из них - **.append()** - добавляет новый элемент в список.
Помните, что **.append()** добавляет элемент в конец списка и никуда более.

Чтобы удалить элемент из списка, используется метод **pop()**, он удаляет **последний** элемент в списке, если вы ничего не укажете в скобках. А если в скобках указать индекс элемента, метод удалит именно этот элемент.

Еще два метода, о которых стоит сказать - это методы сортировки и обратного порядка. В частности иногда я буду просить вас отсортировать списки. Вы можете попытаться сделать это руками, но я дам вам список длиной в 1000 элементов и вы будете страдать (я - нет, просто заставлю переделать).

```
a = [95, 86, 53, 65, 76, -66, 82, 84, -45, 85]
a.sort()
```

Чтобы отсортировать список используется метод **.sort()**. В данном случае у меня был случайный набор чисел, но я захотел получить отсортированный список.

```
a = [95, 86, 53, 65, 76, -66, 82, 84, -45, 85]
a.sort()
a.reverse()
```

Несложно догадаться, что метод **.reverse()** разворачивает последовательность в

обратном порядке.

Заполнение списка

```
from random import randint

a = []
for i in range(10):
    a.append(randint(-100, 100))

print(a)
```

Чтобы наполнить список значениями, мы будем использовать циклы и модуль **random**.

Функция **randint()** используется для генерации случайных чисел в диапазоне, заданном аргументами.

В примере я наполнил список 10 случайными числами в диапазоне от -100 до 100.

Методы списков

Метод	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет перед i-ым элементом значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

Словари

```
shop = {
    'milk': 0.99,
    'banana': 3.45,
    'ham': 2.18,
    'cheese': 3.92,
}
```

Словарь представляет собой набор пар **key: value**, заключение в фигурные скобки. И используется для организации мгновенного доступа к элементам (значениям) по их именам (ключам). Ключи и значения могут иметь разные типы данных. Более того, значением к ключу может быть **список** или **кортеж**.

В словарях нет индексов, обращение происходит по ключам: **shop['milk'] -> 0.99**.

Методы

Методов у словарей не так много. В основном они необходимы для удобного перебора словаря.

```
shop = {
    'milk': 0.99,
    'banana': 3.45,
}

for key in shop:
    print(key)
    # milk banana

for value in shop.values():
    print(value)
    # 0.99 3.45

for key, value in shop.items():
    print(key, '-', value)
    # milk-0.99 banana-3.45
```

Задачи

1. Напишите программу для суммирования всех элементов в списке.
2. Напишите программу для умножения всех элементов в списке.
3. Напишите программу, которая получает наибольшее число из списка.
4. Напишите программу для удаления дубликатов из списка.
5. Напишите сценарий для проверки, существует ли уже данный ключ в словаре.
6. В одномерном числовом массиве D длиной n вычислить сумму элементов с нечетными индексами. Вывести на экран массив D, полученную сумму.
7. В массиве действительных чисел все нулевые элементы заменить на среднее арифметическое всех элементов массива.
8. Дан массив целых чисел. Найти сумму элементов с четными номерами и произведение элементов с нечетными номерами. Вывести сумму и произведение.
9. Дан одномерный массив из 15 элементов. Элементам массива меньше 10 присвоить нулевые значения, а элементам больше 20 присвоить 1. Вывести на экран монитора первоначальный и преобразованный массивы в строчку.
10. Найти наименьший нечетный элемент списка и вывести его на экран.
11. Даны массивы A и B одинакового размера 10. Поменять местами их содержимое и вывести вначале элементы преобразованного массива A, а затем — элементы преобразованного массива B.

Функции и процедуры

Подпрограмма - это именованный фрагмент программы, к которому можно обратиться из другого места программы

Подпрограммы делятся на две категории: процедуры и функции.

Процедуры

```
def имя_процедуры(Список_параметров):
    Система команд
```

Рассмотрим синтаксис процедуры:

Для определения процедуры используется ключевое слово **`def`**, затем указывается имя процедуры и в скобках её формальные параметры, если они присутствуют. После ставится двоеточие и со следующей строки с отступом в **4 пробела** указываются команды.

Процедура — вспомогательный алгоритм, выполняющий некоторые действия.

Процедура должна быть определена к моменту её вызова. Определение процедуры начинается со служебного слова **`def`**.

Вызов процедуры осуществляется по ее имени, за которым следуют круглые скобки, например, **`print()`**.

В одной программе может быть сколько угодно много вызовов одной и той же процедуры.

Использование процедур сокращает код и повышает удобочитаемость.

Процедура с параметрами

Как используются в Python параметры процедуры, рассмотрим на примере.

Пример.

Написать процедуру, которая печатает почтовый адрес для заполнения формы на aliexpress:

```
def address(country, city, zip,
            street, apt, name,):
    print(f'''
    {country}, {city}, {zip}
    {street} st., apart. {apt}
    {name}''')

address('Belarus', 'Minsk', 220080,
        'Karandasheva', '12', 'Vasily Schkryabin')
```

Процедуры и функции работают с пространствами имен.

Глобальная переменная — если ей присвоено значение в основной программе (вне процедуры).

Локальная переменная (внутренняя) известна только на уровне процедуры, обратиться к ней из основной программы и из других процедур нельзя.

Параметры процедуры — локальные переменные.

Функции

Функция - подпрограмма, к которому можно обратиться из другого места программы.

Для создания функции также используется ключевое слово def, после которого указывается имя и список аргументов в круглых скобках. Тело функции выделяется также как тело условия (или цикла): четырьмя пробелами.

Часть функций языка Python являются встроенными функциями, которые обеспечены синтаксисом самого языка. Например, **`int()`**, **`input()`**, **`randint()`**.

Рассмотрим пример создания пользовательских функций.

Пример 1.

Вычислить сумму цифр числа.

```
def sumD(n):  # определение функции с параметром
    summ = 0
    while n != 0:
        summ += n % 10
        n = n // 10
    return summ  # возврат значения функции

print(sumD(1234))  # 10
```

Аннотация типов

Вы можете аннотировать типы данных параметров и возвращаемых значений.

```
def clear_date(date: str) -> tuple:
    c_date, c_time = date[:10], date[13:]
    return c_date, c_time


a, b = clear_date('05/12/2012 Z 13:43:19')
print(f'Date: {a}, time {b}')
```

Вам может показаться, что здесь я указываю тип данных, которые функция может принимать и возвращать. Но это не строгое ограничение. Я могу вписывать любые типы данных, при этом получая лишь уведомления о несоответствии типа данных. Такой способ называется **аннотацией типов**.

У аргументов типы аннотируются через двоеточие. В примере я указываю дословно: параметр **`date: предполагается_строка`**. Возвращаемое значение аннотируется после закрывающей скобки с использованием символов **–>** (на выходе получаем) и **`типа данных`**.

Параметры по умолчанию

Для того, чтобы создавать параметры по умолчанию, вы должны после определения параметра поставить знак принадлежности и вписать туда то самое значение. В этом случае программа не будет ругаться, если вы не передадите функции необходимое количество аргументов.

```
def spam(x: int, y: int = 0, z: int = 0) -> int:
    return 100 * x + 10 * y + 1 * z

print(spam(7, 8, 7))  # 787
print(spam(4, 5))  # 450
```

В случае, если вы хотите аннотировать тип параметров, сначала пишется аннотация, а только потом проставляется значение по умолчанию.

Добавление значений по умолчанию без использования аннотаций в примере 10.1

```
def spam(x, y = 0, z = 0):
    return 100 * x + 10 * y + 1 * z

print(spam(7, 8, 7))  # 787
print(spam(4, 5))  # 450
```

ПРИМЕР 10.1

Важно! Аргументы по умолчанию должны всегда идти последними. Сначала обязательные аргументы (которые необходимо передавать), а только затем аргументы по умолчанию. Иначе получите ошибку **`SyntaxError`**.

Именованные параметры

Когда вы передаете параметры функции, они идут в строгой очерёдности. Первое значение в вызове функции признается первому параметру в определении функции и т.д. Потому что параметры функции **позиционные**, то есть для них важен порядок передачи. Но я имею право нарушать их порядок, если я явно указываю имена параметров.

```
def spam(x: int, y: int = 0, z: int = 0) -> int:
    return 100 * x + 10 * y + 1 * z

print(spam(3, 2, 1))  # 321
print(spam(z=2, x=5, y=8))  # 582
```

Такой вариант присваивания называют именованными параметрами. Как видите, порядок нарушен, но я получаю именно то, что планировал получить, потому что я указал имена параметров, которым я назначаю числа.

Неопределенное количество параметров

Задача

Функции передается неизвестное количество чисел. Необходимо найти их сумму.

Решение

Для решения задачи с числами необходимо научить функцию принимать любое количество аргументов. Для этого необходимо поставить перед аргументом символ * и наблюдать.

```
def summ_all(*nums: int) -> int:
    summary = 0
    for number in nums:
        summary += number
    return summary


print(summ_all(2))  # 2
print(summ_all(2, 3, 5, 6, 1))  # 17
print(summ_all(5, 6, 2, 34, 67, 12, 3, 4, 9))  # 142
```

Задачи

1. а) Составить программу для вычисления площади разных геометрических фигур.
   б) Даны 3 различных массива целых чисел (размер каждого не превышает 15). В каждом массиве найти сумму элементов и среднеарифметическое значение.
2. а) Напишите функцию для суммирования всех чисел в списке
   б) Напишите функцию для нахождения максимума из трех чисел
3. а) Напишите функцию для умножения всех чисел в списке
   б) Напишите функцию для вычисления факториала числа

(неотрицательное целое число). Функция принимает число в качестве аргумента. Использовать функции из math и numpy запрещено.

Для справки: факториалом называется произведение всех натуральных чисел от 1 до n включительно. `3! = 1 * 2 * 3 = 6`

4. а) Напишите функцию для проверки, попадает ли число **n** в заданный диапазон **a-b**. Числа **a, b, n** принимаются как аргументы.
   б) Напишите функцию, которая принимает список из чисел и возвращает новый список с уникальными элементами первого списка (то есть убрать дубликаты из списка и поместить НЕ повторяющиеся элементы в новый список).

5. а) Напишите функцию, которая принимает число в качестве параметра и проверяет, является ли число простым или нет. Простое число - это натуральное число больше 1, которое не имеет положительных делителей, кроме 1 и самого себя.
   б) Напишите функцию Python для создания и печати списка, в котором значения представляют собой квадрат чисел от 1 до 30 (оба включены)

6. а) Напишите функцию, которая принимает любое положительное целое число, а затем возвращает его цифры в отсортированном виде от большего к меньшему. Ввод: 13262, Вывод: 63221
   б) Напишите функцию, которая добавляет правильное окончание к слову “корова” в зависимости от числа, которое передается в виде аргумента. Например: 33 коров**Ы**, 21 коров**А**, 26 коров.

7. а) Дано действительное положительное число **a** и целое число **n**. Вычислите $\mathbf{a^n}$. Решение оформите в виде функции **`power(a, n)`**. Стандартной функцией и оператором возведения в степень пользоваться нельзя.
   б) Научите задачу а работать с отрицательной степенью.

8. а) Напишите функцию, принимающую три аргумента - Год, Месяц, День. Если такая дата существует в нашем календаре, вернуть True, иначе вернуть False. (буду докапываться, если засунете в функцию if…else) Високосные года разрешаю не учитывать при проверке.
   б) Напишите функцию, принимающую 1 аргумент — число от 0 до

1000, и возвращающую True, если оно простое, и False - иначе. (буду докапываться, если засунете в функцию if…else)

9. а) Напишите функцию, принимающую 1 аргумент — сторону квадрата, и возвращающую 3 значения: периметр квадрата, площадь квадрата и диагональ квадрата.
   б) Напишите функцию, принимающую 3 аргумента - длину массива и диапазон, и возвращающую массив с заданным количеством случайных чисел в диапазоне.
10. а) Напишите функцию для перевода двоичного числа в десятичное. Использовать функцию **`int(n, 10)`** нельзя.
    б) Напишите функцию для перевода десятичного числа в довичное. Использоваться функцию **`bin()`** нельзя.
11. а) Напишите функцию, принимающую 3 аргумента - два числа и знак математической операции, и возвращает результат в виде решенного примера. Внимание! Допускается вводить только одну операцию за раз - 1 + 2, не более.
    б) Написать функцию, которая принимает 1 аргумент - номер месяца - и возвращает **название сезона**, к которому относится этот месяц. Например, **передаем 2**, на выходе получаем **'Зима'**.