Python ЛР 2. Часть А. Процедуры и Функции.

❖ *Подпрограммы прежде всего необходимы в ситуации, когда в разных частях программы необходимо выполнять одни и те же действия несколько раз. В таком случае повторяемые операторы оформляются в виде функции или процедуры, к которой можно обращаться и вызывать ее выполнение из разных частей программы.*

❖ *В python существует два вида реализации подпрограмм: процедуры и функции.*

Рассмотрим синтаксис процедуры на примере:
Пример: создать процедуру для вывода сообщения об ошибке. Запрашивать у пользователя ввести положительное число, в случае ввода отрицательного числа, вызывать процедуру для вывода сообщения об ошибке.

✍ Решение:

```
def Err():  # определение процедуры
  print ( "Ошибка: неверные данные" )
n = int ( input('введите положительное число') )
if n < 0:
  Err() # вызов процедуры
```

- Процедура — вспомогательный алгоритм, выполняющий некоторые действия.
- Это поименованный фрагмент программы, который можно вызвать.
- 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, **Err()**.
- 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 60 раз указанный символ (введенный с клавиатуры), каждый с новой строки.

✍ Решение:

```
1 def printChar(s):
2   for i in range(60):
3     print (s)
4 sim = input('введите символ')
5 printChar(sim)
```

- *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. В программе **s** — локальная переменная.

Задание **Python** **3_0:**
Создать процедуру, которая вычисляет разность двух вводимых пользователем числа. Выполнить задание двумя способами: 1) процедура без параметров: числа — глобальные переменные, определенные в основной программе; 2) процедура с параметрами: числа — параметры процедуры.

ЛОКАЛЬНЫЕ И ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

Примеры использования локальных и глобальных переменных:

```
1 x = 3 # глобальная переменная
2 def pr(): # процедура без параметров
3   print (x) # вывод значения глобальной переменной
4 pr()
```

```
1 x = 3  # глобальная переменная
2 def pr(a): # процедура с параметром
3   a = 4 # локальная переменная
4   print (a) # 4
5 pr(x) # передача параметра глобальной переменной (3)
```

 Существует возможность изменить значение глобальной переменной (не создавая локальную). В процедуре с помощью слова **global**:

```
1 x = 3 # глобальная переменная
2 def pr(): # процедура без параметров
3   global x
4   x = 1
5   print (x) # вывод измененного значения глобальной переменной (1)
6 pr(x)
```

Задание **Python** **3_1:**
Напишите процедуру, которая выводит на экран в столбик все цифры переданного ей числа, начиная с последней:

число: 4673

результат:

3

7

6

4

Задание **Python** **3_2:**
Напишите процедуру, которая выводит на экран все делители переданного ей числа (в одну строчку).

Задание **Python** **3_3:**
Составить программу с процедурой для вычисления степени числа (входные параметры: число и степень).

Задание Python 3_4:
Напишите процедуру, которая принимает параметр – натуральное число **N** – и выводит первые *N* чисел Фибоначчи.

Функции

Часть функций языка Python являются встроенными функциями, которые обеспечены синтаксисом самого языка. Например, *int, input, randint*.
Рассмотрим пример создания пользовательских функций.

Пример:
Вычислить сумму цифр числа.

```
1 def sumD(n): # определение функции с параметром
2   sum = 0
3   while n!= 0:
4     sum += n % 10
5     n = n // 10
6   return sum # возврат значения функции
7 # основная программа
8 print (sumD(1075)) # вызов функции с параметром
```

- Функция — это поименованный фрагмент программы, который можно вызвать.
- Как и процедура, функция должна быть определена к моменту её вызова (служебное слово **def**).
- Функция в отличии от процедуры возвращает значение.
- Для возврата значения функции используется оператор **return**.
- Вызов функции осуществляется по ее имени и обычно сопровождается выводом значения.

Задание Python 3_5:
Напишите функцию, которая вычисляет количество цифр числа.

Задание Python 3_6:
Напишите функцию, которая вычисляет факториал натурального числа N.

Python ЛР 2. Часть Б. Массивы (списки)

Массивы (списки) в Питоне, как и в других языках программирования, — это определенное количество элементов одного типа, которые имеют общее имя, и каждый элемент имеет свой индекс — порядковый номер.

L - список целых чисел:

L = [25, 755, -40, 57, -41]

Однако, в языке Python отсутствует такая структура, как «массив». Для работы с массивами используются *списки*.

Списки являются упорядоченными последовательностями, которые состоят из различных объектов (значений, данных), заключающихся в квадратные скобки [ ] и отделяющиеся друг от друга с помощью запятой.

```
list('текст') # ['т', 'е', 'к', 'с', 'т']
```

Создание списков на Python

Создать список можно несколькими способами. Рассмотрим их.

1. Получение списка через присваивание конкретных значений

- Так выглядит в коде Python пустой список:

```
s = []  # Пустой список
```

- Примеры создания списков со значениями:

```
l = [25, 755, -40, 57, -41]   # список целых чисел
l = [1.13, 5.34, 12.63, 4.6, 34.0, 12.8]   # список из дробных чисел
l = ["Sveta", "Sergei", "Ivan", "Dasha"]   # список из строк
l = ["Москва", "Иванов", 12, 124]   # смешанный список
l = [[0, 0, 0], [1, 0, 1], [1, 1, 0]]   # список, состоящий из списков
l = ['s', 'p', ['isok'], 2] # список из значений и списка
```

- Списки можно складывать (конкатенировать) с помощью знака «+»:

```
l = [1, 3] + [4, 23] + [5]

# Результат:
# l = [1, 3, 4, 23, 5]
```

2. Списки при помощи функции List()

Получаем список при помощи функции **List()**

```
l = list ('spisok')  # 'spisok' - строка
print(l) #['s', 'p', 'i', 's', 'o', 'k'] - результат - список
```

3. Создание списка при помощи функции Split()

- Используя функцию **split** в Питон можно получить из строки список. Рассмотрим пример:

```
stroka ="Hello, world"  # stroka - строка
lst=stroka.split(",") # lst - список
lst # ['Hello', ' world']
```

4. Генераторы списков

- В python создать список можно также при помощи генераторов, — это довольно-таки новый метод:
  - Первый простой способ.

  Сложение одинаковых списков заменяется умножением:

```
# список из 10 элементов, заполненный единицами
l = [1]*10
# список l = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

  - Второй способ сложнее.

```
l = [i for i in range(10)]
# список l = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

  или такой пример:

```
c = [c * 3 for c in 'list']
print (c) # ['lll', 'iii', 'sss', 'ttt']
```

  Пример:
  Заполнить список квадратами чисел от 0 до 9, используя генератор списка.

  ✍ Решение:

```
l = [i*i for i in range(10)]
```

  еще пример:

```
l = [(i+1)+i for i in range(10)]
print(l) # [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
```

  - Случайные числа в списке:

```
from random import randint
l = [randint(10,80) for x in range(10)]
# 10 чисел, сгенерированных случайным образом в диапазоне (10,80)
```

Задание Python 4_1:

Создайте список целых чисел от -20 до 30 (генерация).

Задание Python 4_2:

Создайте список целых чисел от -10 до 10 с шагом 2 (генерация list).

Задание Python 4_3:

Создайте список из 20 пятерок (генерация).

Задание Python 4_4:

Создайте список из сумм троек чисел от 0 до 10, используя генератор списка (0 + 1 + 2, 1 + 2 + 3, …).

Ввод списка (массива) в языке Питон

- Для ввода элементов списка используется цикл range:

```
1 for i in range(N):
2   print ( "L[", i, "]=", sep = "", end = "" )
3   L[i] = int( input() )
```

- Более простой вариант ввода списка и его вывода:

```
L = [ int(input()) for i in range(N) ]
```

Функция *int* здесь используется для того, чтобы строка, введенная пользователем, преобразовывалась в целые числа.

- Как уже рассмотрено выше, список можно выводить целым и поэлементно:

```
# вывод целого списка (массива)
print (L)

# поэлементный вывод списка (массива)
for i in range(N):
  print ( L[i], end = " " )
```

Задание Python 4_5:

Заполните массив элементами арифметической прогрессии. Её первый элемент, разность и количество элементов нужно ввести с клавиатуры.

* Формула для получения n-го члена прогрессии: $a_n = a_1 + (n-1) * d$

Задание Python 4_6:

Заполните массив случайными числами в диапазоне 20..100 и подсчитайте отдельно число чётных и нечётных элементов.

Python ЛР 2. Часть B. Строки.

Немного о строках

- Строки в Питоне — это упорядоченная последовательность символов. Строка может заключаться как в кавычки, так и в апострофы:

```
S = 'Dr"s'
S = "Dr's"
```

- Строка считывается со стандартного ввода функцией **input()**:

```
a=input()
print(a)
```

ОПЕРАЦИИ СО СТРОКАМИ

- Для двух строк определена **операция сложения (конкатенации)**, также определена **операция умножения строки на число**:

```
a="па"
b="рад"
print(a+b) # парад
```

```
a="кар"
print (a*4) # каркаркаркар
```

Работа, как с массивами (индексация начинается с *0*):

```
a="парад"
print (a[2]) # p
```

- Длина строки – функция **len()**:

```
a="парад"
print (len(a)) # 5
```

Срезы

Оператор извлечения среза из строки: `[X:Y]`
`X` – это индекс начала среза, а `Y` – его окончания

```
tday = 'morning, afternoon, night'
tday[0:7] # 'morning'
```

Рассмотрим примеры того, как используются в Python срезы:

```
s = 'spameggs'
s[3:5]  # 'me'

s[2:-2]  # 'ameg'

s[-4:-2]  # 'eg'

s[:6]  # 'spameg'

s[1:]  # 'pameggs'

s[:]  # 'spameggs'
```

Можно задать шаг:

```
s = 'spameggs'
s[::-1]  # 'sggemaps'
s[3:5:-1]  # ''
s[2::2]  # 'aeg'
```

Пример:
Извлеките из строки символы с индексами кратными трем.

✍ Решение:

Задание можно выполнить, используя цикл (сложный вариант решения):

```
s = 'spameggs'
x=3
l=len(s)//3
for i in range(l):
  print(s[x:x+1:3]) # m g
  x+=3
```

Для решения можно использовать просто срез:

```
s = 'spameggs'
print(s[1::3])
```

Задание Python 5_1:
Извлеките из строки следующие срезы:
— первые восемь символов;
— четыре символа из центра строки;
— пять символов с конца строки.

Задание Python 5_2:

Дана строка длиной N. Вывести символы строки в обратном порядке. (Не использовать цикл).

Задание Python 5_3:

Дана строка длиной N (N — четное число). Вывести символы с четными номерами в порядке возрастания их номеров:

a2, a4, a6, ... an

Условный оператор не использовать.

Задание Python 5_4:

Дана строка длиной N. Вывести сначала символы с четными номерами (в порядке возрастания номеров), а затем — символы с нечетными номерами (также в порядке возрастания номеров):

a2, a4, a6,...a1, a3, a5...

Условный оператор не использовать.

Методы строк

Строки, как объекты Python, обладают методами (т.е. функциями, которые выполняют сами объекты).

join(str) — *Соединение строк* из последовательности str через разделитель, заданный строкой

```
s="hello"
s1="-".join(s)
s1 # 'h-e-l-l-o'
```

s1.count(s[, i, j]) — количество вхождений подстроки s в строку s1. Результатом является число. Можно указать позицию начала поиска *i* и окончания поиска *j*:

```
s1="abrakadabra"; s1.count('ab') # 2
s1.count('ab',1) # 1
s1.count('ab',1,-3) # 0 , т.к. s1[1:-3]='brakada'
```

s1.find(s[, i, j]) — определяется позиция первого (считая слева) вхождения подстроки s в строку s1. Результатом является число. i и j определяют начало и конец области поиска:

```
s1="abrakadabra"; s1.find('br') # 1
```

s1.replace(s2,s3[, n]) — создаётся новая строка, в которой фрагмент (подстрока) s2 исходной строки заменяется на фрагмент s3. Необязательный аргумент n указывает количество замен:

```
s1="breKeKeKeKs"; ss=s1.replace('Ke','XoXo',2)
ss # breXoXoXoXoKeKs
```

Задание Python 5_5:
Преобразовать дату в «компьютерном» представлении (системную дату: **2016-03-26**) в «российский» формат, т. е. день/месяц/год (например, **26/03/2016**). Известно, что на год выделено всегда 4 цифры, а на день и месяц – всегда 2 цифры.
Примечание:

- Использовать строковые функции языка.
- Функциями работы с датами и временем «заведует» в Python datetime модуль, а непосредственно для работы с датами используется объект date и его методы.

Подсказка:

```
from datetime import date
# Получаем текущую дату
d1=date.today()
# Преобразуем результат в строку
ds=str(d1)
```

Задание Python 5_6:

Ввести адрес файла и «разобрать» его на части, разделенные знаком ‘/’. Каждую часть вывести в отдельной строке.
Например: c:/изображения/2018/1.jpg
Результат:

```
c:
изображения
2018
1.jpg
```

Задание Python 5_7:

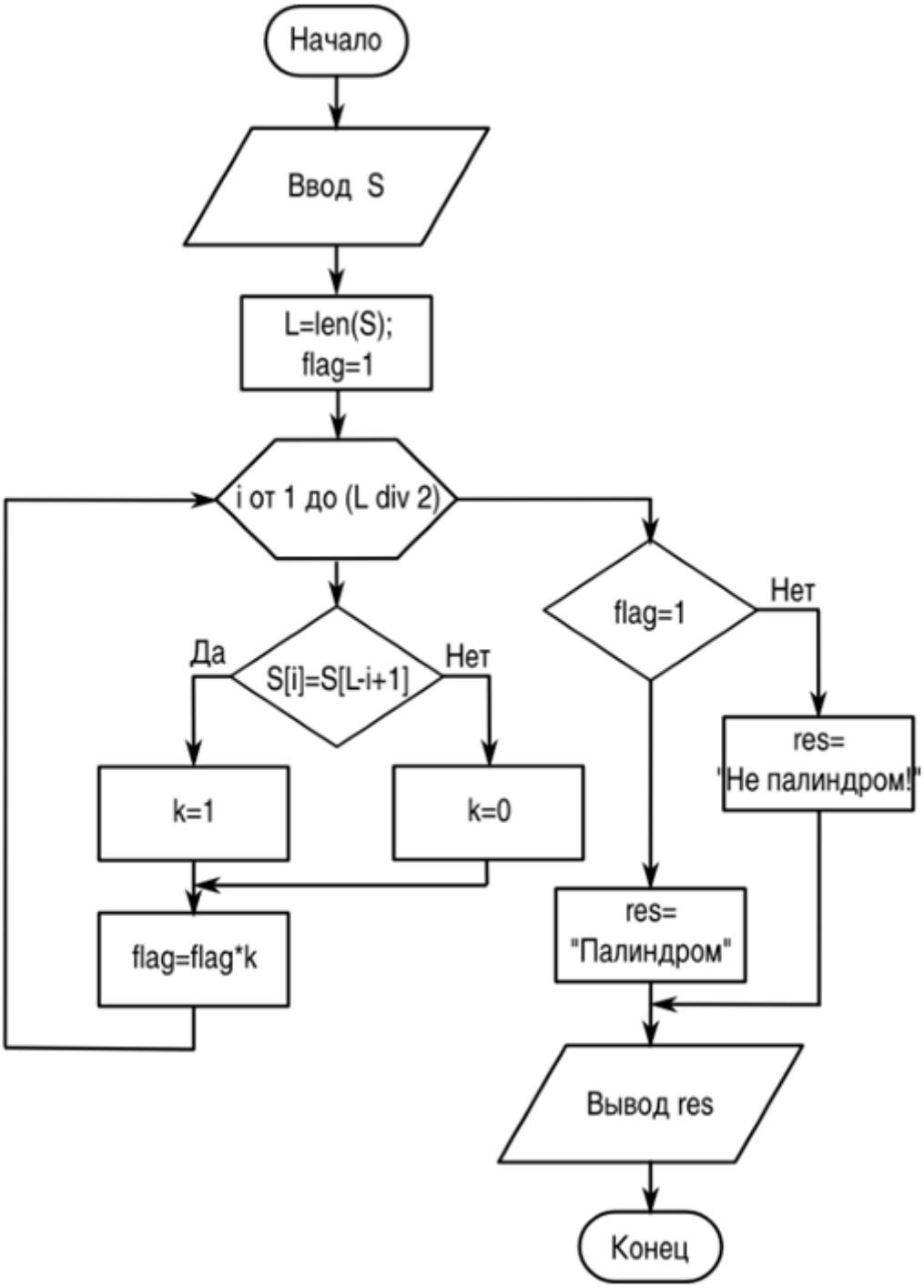
Ввести строку, в которой записана сумма натуральных чисел, например, ‘1+25+3’. Вычислите это выражение. Использовать строковые функции языка.

Задание Python 5_8:
Определить, является ли введённая строка палиндромом («перевёртышем») типа *ABBA*, *kazak* и пр.
Примечание:

если s='1234', то

S[-1]='4', s[-2]='3'….

Для решения используйте алгоритм, изображенный на блок-схеме:



Самостоятельная работа:

1 вариант:

Функции:

1. Описать функцию *CircleS(R)* вещественного типа, находящую площадь круга радиуса *R*. С помощью этой функции найти площади трех кругов с заданными радиусами. Площадь круга радиуса *R* вычисляется по формуле $S=\pi*R^2$. В качестве значения *Пи* использовать *3.14*.

Процедуры:

2. Напишите процедуру, которая выводит цифры переданного ей четырехзначного числа, кратные трем.

Списки:

3. Создайте список, соответствующий числовому ряду, построенному по следующему принципу:

1+2+3 2+3+4 3+4+5 ... 8+9+10

т.е. результат:

[6, 9, 12, 15, 18, 21, 24, 27]

2 вариант:

Функции:

1. Описать функцию *TriangleP(a, h)*, находящую периметр равнобедренного треугольника по его основанию *a* и высоте *h*, проведенной к основанию. С помощью этой функции найти периметры трех треугольников, для которых даны основания и высоты (периметр = сумме длин всех сторон). Для нахождения боковой стороны b треугольника использовать теорему Пифагора: $b^2=(a/2)^2+h^2$.

Процедуры:

2. Напишите процедуру, которая выводит произведение цифр переданного ей четырехзначного числа.

Списки:

3. Создайте список, соответствующий числовому ряду, построенному по следующему принципу:

1*3 2*4 3*5 ... 8*10

т.е. результат:

[3, 8, 15, 24, 35, 48, 63, 80]

Срезы:
Общее:

4. Дана строка. Вывести символы с нечетными номерами в порядке убывания их номеров (0-й символ тоже рассмотерть).

Например, для строки:

```
"Друзья!", результат:  "язр"


"Друзья", результат:  "язр"
```

Форматирование строк

Python включает форматирование строк. Данное понятие подразумевает подстановку какого-либо шаблона в определенное место (или в определенные позиции) текста. Подстановка происходит, что называется, «на лету».

Рассмотрим пример использования в коде на Python метода **format**:

- Одна подстановка:

```
'Hello, {}!'.format('Vasya') # 'Hello, Vasya!'
```

Аргументом метода является текст-подстановка, который при исполнении программы подставляется на место фигурных скобок.

- Несколько подстановок:

```
'{0}{1}{0}'.format('abra', 'cad') # 'abracadabra'
```

Подстановки нумеруются, аргументы метода format заполняют позиции для подстановок согласно их порядковым номерам, указанным в фигурных скобках.

- другой вариант форматирования с множественными подстановками:

```
'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.81W')
'Coordinates: 37.24N, -115.81W'
```

Аргументы метода format заполняются согласно указанным именам заполнителей.

У метода format есть больше возможностей, которые выходят за рамки рассмотрения темы на данном уроке.

Задание Python 5_9:

Допустим, есть какое-то объявление и несколько разных людей, которым нужно это объявление отправить. Для этого создается заготовка с содержанием объявления, внутри которого есть ряд изменяющихся параметров: имена людей и названия событий. Вывести один вариант

итогового объявления на экран. Для задания имен и названий использовать форматирование при помощи метода format.

Используйте шаблон объявления и ориентируйтесь по цветам:

Красным – массивы.

Коричневым – числовая переменная.

Уважаемый (ая), Иван Иванович!

Приглашаем Вас на день открытых дверей.

Дата события: 1 мая.

С уважением, Василий.

* Примечание:

- Вывести пять объявлений.
- Для имен создать массив (список), подстановку элементов в объявление осуществлять в цикле.
- Даты меняются от 1 до 5.

Python ЛР 2. Часть Г. Операции соединения и повторения.

- *Списки — это изменяемые последовательности* в отличие от строк.
- Представим строку как объект в памяти, в этом случае, когда над строкой выполняются операции конкатенации и повторения, то сама строка не меняется, но в результате выполнения операции в другом месте памяти создается другая строка.

  В строку нельзя добавить новый символ или удалить существующий, не создав при этом новой строки.

  Пример:
  Так, например, в Питоне нельзя переприсваивать значение для отдельных символов строки.
  Программа выдаст ошибку!

```
s="aaa";
s[1]="b";
print(s1)
```

  Изменять строку можно только, работая с ней, **как с объектом** (метод replace, например):

```
s1="breKeKeKeKs";
s1=s1.replace('Ke','XoXo',2)
s1 # breXoXoXoXoKeKs
```

  Что касается списков, то при выполнении операций другие списки могут не создаваться, при этом *изменяется непосредственно оригинал*.
  Из списков можно **удалять и добавлять новые элементы.**

- Операция конкатенации:

```
[33, -12, 'may'] + [21, 48.5, 33] # [33, -12, 'may', 21, 48.5, 33]
```

  или так:

```
a=[33, -12, 'may']
b=[21, 48.5, 33]
print(a+b)# [33, -12, 'may', 21, 48.5, 33]
```

- Операция повторения:

```
[[0,0],[0,1],[1,1]] * 2 # [[0, 0], [0, 1], [1, 1], [0, 0], [0, 1], [1, 1]]
```

Пример:
Для списков операция переприсваивания значения отдельного элемента списка **разрешена!**:

```
a=[3, 2, 1]
a[1]=0;
print(a) # [3, 0, 1]
```

Можно!

Задание 6_1:

Ввести строку, в которой записана сумма натуральных чисел, например, ‘1+25+3’. Вычислите это выражение. Работать со строкой, как со списком.
Начало программы:

```
s=input('введите строку')
l=list(str(s));
```

Другие операции над списками при помощи функций

```
a=[1,7,3,88,33]
a.sort() #[1,3,7,33,88]  - сортировка
a.reverse() #[88,33,7,3,1]  - обратная сортировка
a.index(7) #2  - индекс элемента
a.clear() #  - очистка списка
len(a) #  - длина списка
sum(a) #  - суммирование элементов
```

Функция join() — соединение элементов через определенный символ:

```
lst=['11','22','33']
lst="-".join(lst)# '11-22-33'
```

Функция split([sep]) — возвращает список подстрок, получающихся разбиением строки а разделителем sep:

```
str="1-2-3-4"
s1=str.split("-") # ['1','2','3','4']
```

Задание 6_2:

Дан список из 5 различных элементов. Используя функции (не использовать цикл), необходимо найти и вывести:

- минимальный и максимальный элементы списка;
- сумму и среднее арифметическое;
- второй минимальный элемент (второй по минимальности).

Начало программы:

```
lst=[4,5,2,3,4]
```

Добавление и удаление элементов списка

- Добавление элемента:

```
>>> a=[]
>>> a.append('444')
>>> a
['444']
```

Пример:
Поиск нечетных элементов в массиве *mas* и копирование их в массив *B*.

✍ Решение:

```
B = []
for x in mas:
    if x % 2 != 0:
        B.append(x)
```

- Удаление элемента:

```
> a.remove('444')
>>> a
[]
```

- Удаление элемента по индексу:

```
>>> del a[0]
>>> a
[]
```

При копировании списков, т.е. присваивании одного списка другому, изменение первого списка влечет за собой изменение второго списка. Так как эти объекты связаны одной областью памяти (ссылка на список).

```
mas1 =[1, 2, 3]
mas2 = mas1 # создается ссылка на список

mas1[0] = 4
print(mas2) #[4, 2, 3]
```

Чтобы создать не ссылку на список, а копию списка можно использовать либо срез либо функцию *copy*.

1.

```
mas2 = mas1[:] # используем срез
```

2.

```
import copy
mas1 =[1,2,3]
mas2 = copy.copy(mas1)
```

Задание 6_3: Проверить, является ли заданное слово палиндромом.
Примечание:

- Пример палиндрома: *казак*, *ABBA*
- Использовать функции.
- Поскольку при присваивании одного списка другому, изменение первого ведет к аналогичному изменению второго списка, то необходимо использовать копию (copy).

Начало программы:

```
import copy
stroka=input('введите слово')
lst=list (stroka) # конвертируем строку в список
```

Генерация случайных чисел

Встроенный модуль Питона random позволяет генерировать псевдослучайные числа.

Модуль **random** включает в себя функцию *random*, которая возвращает действительное число в диапазоне от *0.0* до *1.0*. Каждый раз при вызове функции возвращается число из длинного ряда.

Пример:

```
import random
for i in range(10):
  x = random.random()
  print (x) # 0.5185207383774904  0.78283351055836  0.23601341583293534  ...
```

Чтобы получить случайное число между *0.0* и верхней границей *high*, просто умножьте *x* на high.

Например, от 0.0 до 15.0:

```
import random
for i in range(10):
```

```
x = random.random()
print (x * 15) # 11.075319687990554  7.152253113207329 ...
```

Для того, чтобы получить псевдослучайное целое число:

```
import random

random.randint(<начало>,<конец>)
```

Для того, чтобы получить псевдослучайное вещественное число:

```
import random

random.uniform(<начало>,<конец>)
```

Еще пример:

```
from random import randint
l = [randint(10,80) for x in range(10)]
```

Задание Python 6_4:

Найдите в массиве все простые числа и скопируйте их в новый массив.

Задание Python 6_5:

Решить задачу поиска среднего значения в списке из N элементов (вводимых в виде строк). Использовать метод добавления элементов списка и суммирования элементов

Цикл for при работе со списками

```
mylist=[1,2,3,4,5]
for item in mylist:
    item = 0
# mylist не меняется!
print(mylist) # [1, 2, 3, 4, 5]
n=5
for i in range(n):
    mylist[i] = 0
# mylist меняется
print(mylist) # [0, 0, 0, 0, 0]
```

В списке чисел проверить, все ли элементы являются уникальными, т.е. каждое число встречается только один раз

✍ Решение:

Комментарии к программе:
Решать данную задачу на языке Python мы будем «классическим» вариантом — брать по очереди элементы списка и сравнивать каждый элемент со стоящими за ним. При первом же совпадении элементов делается вывод, что в списке есть одинаковые элементы, и работа программы завершается.

Для выхода из цикла будем использовать метод quit()

```
import random
m = 6
mass=[]
k=0
j=0
for i in range(m):
  mass.append(random.randint(-10,10))
  print(mass[i]) # -10 0 -8 0 -10 1
for i in mass:
  k=k+1
  for j in range(k,m): # j = 0 -8  0 -10 -> quit
          if (i==mass[j]): # -10==0  -10==-8  -10==0  -10==-10 -> quit
                print('yes')
                quit()
```

Задание Python 6_6:
Определить индексы элементов массива (списка), значения которых принадлежат заданному диапазону (т.е. не меньше заданного минимума и не больше заданного максимума)

```
 -5  9  0  3 -1 -2  1  4 -2 10  2  0 -9  8 10 -9  0 -5 -5  7
min= 5
max= 15
кол-во:  5
индексы:  [1, 9, 13, 14, 19]
```

* в результате получили индексы элементов, значения которых находятся в диапазоне [5,15]

Алгоритм:

1. Заполнить список (массив) случайными числами
2. Запросить для ввода минимум и максимум диапазона
3. Найти индексы элементов, значения которых входят в диапазон. Добавлять найденные индексы к новому списку
4. Вывести общее число найденных индексов (функция len()) и отдельно все индексы

Задание Python 6_7:

Дополнить предыдущую программу следующим:
После того, как элемент с подходящим значением добавлен в новый список – удалять его из исходного списка

* Использовать цикл while , функцию len()