Информатика. Python. Базовый синтаксис.

Python — это язык программирования. Как любой язык программирования он устроен как некая последовательность команд, которые выполняются и приводят к какому-то результату.

Рассмотрим базовый синтаксис языка Python – операции присвоения, ввода и вывода, сравнение чисел, ветвление, циклы, работу с массивами, функции.

Присвоение и вывод:

```
x = 5
print(x)
```

Данная программа выведет число 5

Арифметические операции:

```
x = 7
y = 3
print(x+y)
```

Программа выведет сумму чисел 7 и 3 — 10

```
x = 7
y = 3
print(x - y)
```

Программа выведет разность чисел 7 и 3 — 4

```
x = 7
y = 3
print(x * y)
```

Программа выведет произведение чисел 7 и 3 — 21

```
x = 7
y = 3
print(x / y) #нецелочисленное деление
```

Программа выведет результат нецелочисленного дедения 7 на 3 — 2.(3)

```
x = 7
y = 3
print(x // y) #целочисленное деление
print(x % y)  #операция взятия остатка
```

Программа выведет результат целочисленного дедения 7 на 3 — 2 и остаток от деления — 1

Операции сравнения:

```
x = 7
y = 3
print(x > y)
```

Программа выведет True – так как x правда больше y

```
x = 7
y = 3
print(x < y)
```

Программа выведет False – так как x не меньше y

Знаки сравнения : меньше или равно, больше или равно, равно и не равно:

```
x = 7
y = 3
print(x <= y)
print(x >= y)
print(x == y)
print(x != y)
```

Данные операции в большей мере нужны в операциях ветвления — для проверки каких-либо условий в программе.

Операции ветвления:

```
x = 7
y = 3
if (x > y): #условие если
    print("Hello!")
else: #иначе
    print("Goodbye!")
```

Ввод:

```
x = input()
print(x)
```

Данная программа принимает на вход как числа, так и строки – результатом команды input является считывание с клавиатуры строки, состоящей из любых символов (т.е. даже если мы введем число, то в x будет храниться просто строка — массив символов, и питоном без дополнительных преобразований не будет считаться за число)

```
x = input()
y = input()
print(x+y)
```

При входных данных 13 и 17 программа выведет 1317, как результат сложения строк. Для сложения чисел 13 и 17 нужно повесить int на каждое из чисел.

встроенная функция int() возвращает целое число в десятичной системе счисления. Также если вторым параметром указать число k, то это будет значить что строка является числом в k-ой системе счисления и мы переводим его в десятичную.

```
x = int(input())
y = int(input())
if (x > y):
    print(x)
else:
    print(y)
```

Данная программа выводит наибольшее число из входных данных

Еще раз input() — позволяет ввести любую строчку. К строчке можно применять строковые методы.

Мы можем разбить строчку на два объекта, разделенных каким-либо символом (split() — разделение. Если в скобках не указан какой-либо символ, то разделяем строку на части до пробела и после):

```
a, b = input().split() #вводим с клавиатуры два числа в строку
print(int(a) + int(b)) #переводим каждые объекты в числа и выводим
```

Для трёх объектов:

```
a, b, c = input().split() #разбили на три объекта
print(int(a) + int(b) + int(c))
```

Если нужно ввести три числа в одну строку и затем работать с ними как с числами, то это можно сделать следующим образом:

```
a, b, c = inpit().split()
a = int(a) #переводим строчки в числа
b = int(b)
c = int(c)
print(a + b + c)
```

Чтобы применить функцию int() ко всем элементам split(), нужно использовать функцию map():

```
a, b, c = map(int, inpit().split())
print(a + b + c )
```

Также способом ввода является массив. Например с помощью генератора : для всех s элементов множества input().split()

```
a, b, c = [int(s) for s in input().split()]
print(a + b + c )
```

Циклы:

Цикл While

```
x = 0
while (x < 100): #пока x < 100 делай
    print(x)
```

Данная программа выводит бесконечное количество нулей. Почему? Потому что ноль всегда меньше 100, а x в программе не изменяется

```
x = 0
while (x < 100): #пока x < 100 делай
    print(x)
    x = x + 5
```

Данная программа выводит числа от 0 до 95 с шагом 5

```
x = 0
while (x < 100): #пока x < 100 делай
    x = x + 5
 print(x)
```

Данная программа выводит число 100

```
x = 0
while (x < 100): #пока x < 100 делай
    print(x*x)
    x = x + 1
```

Программа, выводящая таблицу квадратов чисел от 0 до 99

```
x = 0
while (x < 100): #пока x < 100 делай
    if (x % 7 ==3) and (x % 2 == 0): #вместо and можно использовать &
        print(x)
    x = x + 1
```

Программа, выводящая все чётные числа, дающие остаток 3 при делении на 7

Задача: выписать все числа от 0 до 100, большие 25, при этом не кратные 17:

```
x = 0
while (x < 100):
    if (x > 25) and (x % 17 != 0):
        print(x)
     x = x + 1
```

Цикл for

```
#for x in ДИАПАЗОН (диапазон обычно задается в виде массива):
#     действия

for x in [0, 1, 2, 3, 4, 5, 6, 7, 8]:
    print(x*x)
```

Программа выведет квадраты чисел от 0 до 8. То же самое, как мы уже знаем, выведет следующая программа:

```
x = 0
while (x < 9):
    print(x*x)
    x = x + 1
```

Чтобы получить диапазон от 0 до 99 нужно использовать range(100). Программа будет проходить по каждому элементу диапазона:

```
#for x in ДИАПАЗОН:
#     действия
# [0, 1, 2, 3, 4, 5, 6, 7, 8, ..., 99] == range(100)
for x in range (100):
    print(x*x)
```

Массивы:

```
a = [10, 12, 100, 35, 43] #задаём массив из 5 чисел
print(a[0]) #выводим нулевой элемент массива
print(a[3]) #выводим третий элемент массива
print(a[2:4]) #выводим со 2 по 4 (не включительно) элементы
print(a[:4]) #выводим со 0 по 4 (не включительно) элементы
print(a[2:]) #выводим со 2 элементы
print(a[1:5:2]) #выводим 1 и 3 элементы
```

Посчитать количество чётных элементов в массиве

```
a = [10, 12, 100, 35, 43] #задаём массив из 5 чисел
counter = 0 #обнуляем счётчик
for i in range(5): #диапазон из  5 элементов массива
```

```
    if (a[i] % 2 ==0): #условие проверки чётности i-того элемента массива
            counter = counter + 1
 #обновляем счётчик, если такое число нашлось
print(counter)
```

Дополнить массив числами

```
a = [] #задали массив a
a.append(36) #помещаем число в конец массива
a.append(45)
a.append(20 + 30)
print(a) #выводим массив
```

Заполнить массив с помощью цикла for числами с клавиатуры

```
a = []
for i in range(10): #диапазон размером 10
    x = int(input()) #считываем число
    a.append(x) #помещаем число x в массив
print(a) #выводим массив
```

Задача посчитать количество нечётных элементов в массиве, а затем заменить все нечётные элементы этим количеством и вывести изменённый массив

```
N = 30 #константа N
a = []
for i in range(N): #диапазон из N чисел
     x = int(input())
     a.append(x) #заполняем массив с клавиатуры
k = 0 #обнуляем счётчик элементов

for i in range(N):
     if (a[i] % 2 != 0): #условие на нечётность i-того элемента массива
          k = k + 1 #прибавляем 1 к счётчику, если такой элемент нашелся

for i in range(N):
     if (a[i] % 2 != 0):
          a[i] = k #заменяем i-тый элемент массива на найденное кол-во

for i in range(N):
     print(a[i]) #выводим изменённый массив
```

Также нередкими ситуации бывают, что в задаче нужно найти пары, тройки и так далее различных чисел, удовлетворяющих какому-то условию. Самым простым способом с точки зрения подобной реализации служит создание 2, 3 и так далее вложенных циклов, позволяющих получить всевозможные пары, тройки и так далее из набора данных.

Программа, выводящая всевозможные пары чисел от 1 до 10

```
for i in range(1, 11):
    for j in range(1, 11):
        print(i, j)
```

Можно заметить, что при подобной реализации будут напечатаны одни и те же пары, но с различной перестановкой чисел (Например, будет пара (1,10) и (10, 1), что в рамках многих задач может являться одинаковой парой). Чтобы решить данную проблему нужно при старте второго цикла указать значение i+1 — в таком случае для каждого первого взятого числа мы будем ставить ему в пару только идущие после него.

```
for i in range(1, 11):
    for j in range(i+1, 11):
        print(i, j)
```

А что делать, если требуется найти максимальный или минимальный элемент в наборе чисел? Если требуется найти максимальное число, то изначально задаем минимальное значение, которое может быть во входных данных (можно и меньше, но не больше!). Почему? Потому что в дальнейшем проходя по числам мы будем смотреть, является ли число больше, чем предыдущее значение максимального элемента. Если да, то заменяем максимальный элемент.

С минимальным тоже самое — только изначально задаем очень большим значением, а затем уменьшаем, если находим число меньшее по значению.

Задача найти максимальный и минимальный элементы в массиве

```
N = int(input())
a = []
for i in range(N):
    x = int(input())
    a.append(x)

maxim = 0
minim = 10000000000

for i in range(N):
    if a[i] > maxim:
        maxim = a[i]
    if a[i] < minim:
        minim = a[i]

print(maxim, minim)
```

Функции:

Функции позволяют разбивать программу на исполняемые фрагменты — для большего удобства использования. К примеру, один фрагмент программы может повторяться более одного раза или подобное визуальное разбиение помогает лучше в смысловом плане отделять фрагменты программы.

Пример функции, которая принимает число на вход и если оно является четным, то просто выводит его, а если нечетным — увеличивает его на 1 и также выводит

```
def name(n):
    if n % 2 == 0:
        return n
    else:
        return n+1

print(name(n))
```

Задача: определить, сколько чисел от 2 до 100 являются простыми (делятся только на 1 и на само себя)

```
def prime(x):
    for i in range(2, x+1):
        if x % i == 0:
            return False
    return True

ans = 0
for i in range(2, 100+1):
    if prime(i):
        ans += 1
print(ans)
```