

Решение задание №5 ЕГЭ по информатике с использование Python

УРОК № 5. ВЫПОЛНЕНИЕ И АНАЛИЗ ПРОСТЫХ АЛГОРИТМОВ.

(базовый уровень, время – 4 мин)

Тема: **ВЫПОЛНЕНИЕ И АНАЛИЗ ПРОСТЫХ АЛГОРИТМОВ.**

Что проверяется:

Формальное исполнение алгоритма, записанного на естественном языке, или умение создавать линейный алгоритм для формального исполнителя с ограниченным набором команд.

1.6.3. Построение алгоритмов и практические вычисления.

1.1.3. Умение строить информационные модели объектов, систем и процессов в виде алгоритмов.

Что нужно знать:

- сумма двух цифр в десятичной системе счисления находится в диапазоне от 0 до 18 (9+9)
- в некоторых задачах нужно иметь представление о системах счисления (могут использоваться цифры восьмеричной и шестнадцатеричной систем счисления)
- **бит чётности** – это дополнительный контрольный бит, который добавляется к двоичному коду так, чтобы количество единиц в полученном двоичном коде стало чётным; если в исходном коде уже было чётное количество единиц, дописывается 0, если нечётное – дописывается 1.
- при добавлении к двоичной записи числа нуля справа число увеличивается в 2 раза
- чтобы отбросить последнюю цифру в двоичной записи, нужно разделить число на 2 нацело (остаток отбрасывается)

ФУНКЦИЯ BIN() В PYTHON

Функция **bin()** в Python преобразует целое число в двоичную строку.

Синтаксис:

```
bin(x)
```

Параметры:

- **x** - [целое число](#)

Возвращаемое значение:

- двоичная строка с префиксом `0b`.

```
Type "copyright",
>>> bin(7)
'0b111'
>>> bin(7)[2:]
'111'
>>>
```

Описание:

Функция **bin()** преобразует [целое число](#) в двоичную строку с префиксом `0b`.

Результатом будет **binary string** - двоичная версия заданного целого числа **x**.

Примеры преобразований чисел в двоичную систему счисления.

```
>>> bin(3)
# '0b11'
>>> bin(-10)
# '-0b1010'
```

Если префикс 0b является НЕ желательным, вы можете использовать [2:]

```
>>> bin(6)
'0b110'
```

и когда нужно стереть '0b', то используем [2:] :

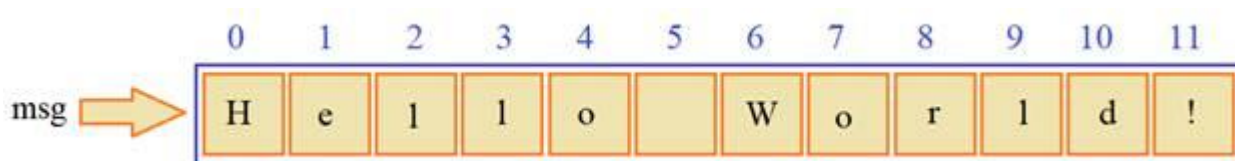
```
>>> bin(6)[2:]
'110'
```

ИНДЕКСЫ И СРЕЗЫ СТРОК

На самом деле в Python строка представляется как упорядоченная коллекция символов. И ключевое слово здесь – «упорядоченная». Это значит, что у каждого символа в строке есть свой порядковый номер – индекс, по которому мы можем его получить. Например, когда мы создаем строку

```
msg = "Hello World!"
```

то формируется следующая коллекция:



Каждый символ имеет свой индекс, начиная с нулевого. Первый символ в Python всегда имеет нулевой индекс.

Для обращения к тому или иному символу используется следующий синтаксис:

<строка>[<индекс>]

Например:

```
msg[0]
msg[6]
```

и так далее. Но, если указать неверный индекс, например:

```
msg[12]
```

то возникнет ошибка. Поэтому здесь следует быть аккуратным и не выходить за пределы этого списка. В частности, последний «рабочий» индекс можно определить с помощью функции len – длины строки:

$\text{lastIndex} = \text{len}(\text{<строка>}) - 1$

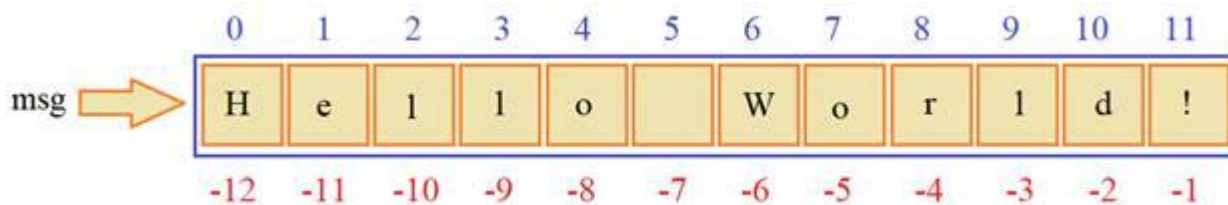
То есть, к последнему индексу мы можем обратиться так:

```
msg[len(msg)-1]
```

Но это не очень удобно. Поэтому разработчики языка Python решили, что отрицательные индексы будут означать движение по строке с конца в начало. И предыдущую запись можно переписать так:

```
msg[-1]
```

Видите? Это намного удобнее. То есть, у строк есть еще такие отрицательные индексы:



Также в Python можно использовать доступ к отдельному символу непосредственно у строкового литерала:

```
"abcd"[1]
```

```
"abcd"[-1]
```

Иногда это бывает удобно.

Срезы

Часто в программировании требуется выбрать не один какой-то символ, а сразу несколько. Для этого используются так называемые срезы. Их работу проще показать на конкретных примерах. Пусть у нас есть наша строка:

```
msg = "Hello World!"
```

и мы хотим выделить последнее слово «World!». Для этого в **квадратных скобках указывается начальный индекс и через двоеточие – конечный**. Если мы запишем все вот в **таком виде**:

```
msg[6:11]
```

то получим результат «World» без восклицательного знака. Дело в том, что последний индекс исключается из интервала, то есть, интервал определяется как [6: 11)

Поэтому, мы должны записать срез так:

```
msg[6:12]
```

Другой пример для выделения символов «llo»:

```
msg[2:5]
```

и так далее. В Python допускается не указывать начальное или конечное значения, или даже, оба из них. Например:

```
msg[:5]
```

выделяет слово «Hello», а вот так:

```
msg[6:]
```

получим «World!».

Метод append() и extend() в Python

Метод append() в Python добавляет элемент в конец списка. Синтаксис метода:

```
list.append(item)
```

Пример 1: Добавление элемента в список

```
list animals = ['cat', 'dog', 'rabbit']
```

```
animals.append('guinea pig')
```

```
print('Updated animals list: ', animals)
```

Выход:

```
list: ['cat', 'dog', 'rabbit', 'guinea pig']
```

Пример № 0. ДЕМО-2022 задания №5

На вход алгоритма подаётся натуральное число N . Алгоритм строит по нему новое число R следующим образом.

1. Строится двоичная запись числа N .
2. Далее эта запись обрабатывается по следующему правилу:
 - а) если число чётное, то к двоичной записи числа слева дописывается 10;
 - б) если число нечётное, то к двоичной записи числа слева дописывается 1 и справа дописывается 01.

Полученная таким образом запись является двоичной записью искомого числа R .

Например, для исходного числа $4_{10} = 100_2$ результатом будет являться число $20_{10} = 10100_2$, а для исходного числа $5_{10} = 101_2$ результатом будет являться число $53_{10} = 110101_2$.

Укажите минимальное число N , после обработки которого с помощью этого алгоритма получается число R , большее, чем 441. В ответе запишите это число в десятичной системе счисления.

```
str1 = []
for N in range(1,310):
    N_b = bin(N)[2:]
    if N%2 == 0:
        N_b = ".join(['10',N_b])
        #N_b = '10'+N_b
    else:
        N_b = ".join(['1',N_b,'01'])
    if int(N_b,2)>441:
        str1.append(N)
print(min(str1))
```

Пример № 1. ДЕМО-2023 задания №5

5 На вход алгоритма подаётся натуральное число N . Алгоритм строит по нему новое число R следующим образом.

1. Строится двоичная запись числа N .
2. Далее эта запись обрабатывается по следующему правилу:
 - а) если сумма цифр в двоичной записи числа чётная, то к этой записи справа дописывается 0, а затем два левых разряда заменяются на 10;
 - б) если сумма цифр в двоичной записи числа нечётная, то к этой записи справа дописывается 1, а затем два левых разряда заменяются на 11.

Полученная таким образом запись является двоичной записью искомого числа R .

Например, для исходного числа $6_{10} = 110_2$ результатом является число $1000_2 = 8_{10}$, а для исходного числа $4_{10} = 100_2$ результатом является число $1101_2 = 13_{10}$.

Укажите минимальное число N , после обработки которого с помощью этого алгоритма получается число R , большее 40. В ответе запишите это число в десятичной системе счисления.

Ответ: _____.

```
for n in range (1,100):
```

```
    b=bin(n)[2:]
```

```
    if b.count('1')%2==0:
```

```
        b='10'+b[2:]+'+0'
```

```
    else:
```

```
        b='11'+b[2:]+'+1'
```

```
        r=int(b,2)
```

```
        if r>40:
```

```
            print(n)
```

```
#Ответ 16
```

То же, с помощью функции:

```
ss = []

def sum(N):
    R = bin(N)[2:]
    if (R.count('1')%2 == 0):
        R = '10' + R[2:]+ '0'
    else:
        R = '11' + R[2:]+ '1'
    return (int(R,2))

for i in range(1,100):
    if sum(i)>40:
        ss.append(i)

print(f {min(ss)}')
```

№ 5899 (Уровень: Сложный)

(Д. Тараскин) Автомат получает на вход трёхзначное число. По этому числу строится новое число по следующим правилам:

- 1) Из цифр, образующих десятичную запись N, строятся все возможные двузначные числа (числа не могут начинаться с нуля).
- 2) Из получившихся двузначных чисел выбираются только те, которые являются простыми.

Каждую цифру трехзначного числа можно использовать ровно столько раз, сколько она встречается в этом числе. К примеру, возьмем число 123. Из него можно составить числа: 12, 13, 21, 31, 23, 32.

Для какого наибольшего N количество выбранных простых чисел будет максимальным?

Решение:

```
def PR(n):
    for i in range(2,n):
        if n%i==0:
            return 0
    return 1

max_l, max_c= 0,0
for i in range(100,1000):
    s = str(i)
    d = [s[0]+s[1],s[0]+s[2],s[1]+s[0],s[2]+s[0],s[2]+s[1],s[1]+s[2]]
    N = {int(h) for h in d if h[0]!='0' and PR(int(h))==1}
    if max_l<=len(N):
        max_l = len(N)
        max_c = i
print(max_c)
```

Когда строк много

Представим, что вы получили много строк в одном списке и теперь надо объединить их в одну. Попробуем решить задачу с помощью плюса:

```
strings = ["Жизнь", 'слишком', 'коротка,', 'программируй', 'на', 'Python']
def join_strs(strs):
    result = ""
    for s in strs:
        result += ' ' + s # а вот и плюс
    return result[1:]

join_strs(strings)>>> 'Жизнь слишком коротка, программируй на Python'
```

```
".join((s1, s2)) # кортеж>>> 'Я люблю программировать на Python'
".join({s1:'a', s2:'b'}) # словарь с s1 и s2 в качестве ключей>>> 'Я люблю программировать на Python'
".join({'a':s1, 'b':s2}) # словарь с s1, s2 в качестве значений>>> 'ab'
```