Практическое занятие №4.

Работа с битовыми полями и регулярными выражениями

При решении следующих задач предлагается выполнять автоматическую проверку синтаксиса кода в соответствии со стандартом PEP8. Такая практика поможет Вам научиться писать код на языке Python сразу без синтаксических ошибок!

In [ ]:
```
# установка библиотеки для проверки кода на соответствие PEP8
pip install flake8 pycodestyle_magic
```

In [ ]:
```
# теперь код будет проверяться на соответствие PEP8
%load_ext pycodestyle_magic
%pycodestyle_on
```

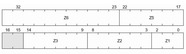
1. Реализовать функцию для кодирования данных, содержащих битовые поля. В решении необходимо использовать побитовые операции. Неиспользуемые поля результата должны содержать нулевые биты.

Входные данные:

Словарь из битовых полей. Значения битовых полей имеют тип: десятичная строка.

Выходные данные:

Шестнадцатиричная строка.



Тесты:

```
>>> main({'z1': '3', 'z2': '2', 'z3': '11', 'z5': '32', 'z6': '525'})
'0x100c53613'

>>> main({'z1': '5', 'z2': '55', 'z3': '2', 'z5': '53', 'z6': '1016'})
'0x1fc6ad35a7'

>>> main({'z1': '3', 'z2': '46', 'z3': '41', 'z5': '29', 'z6': '387'})
'0xc3be3773'

>>> main({'z1': '3', 'z2': '47', 'z3': '51', 'z5': '40', 'z6': '550'})
'0x11350d7b'
```

2. Реализовать функцию для кодирования данных, содержащих битовые поля. В решении необходимо использовать побитовые операции.

Входные данные:

Кортеж из битовых полей в порядке от младших бит к старшим. Значения битовых полей имеют тип: целое.

Выходные данные:

Целое.

Тесты:

```
>>> main({376, 159, 1, 0, 1, 21})
361054072

>>> main({130, 60, 1, 11, 1, 52})
886864002

>>> main({396, 444, 0, 14, 0, 33})
561215884

>>> main({110, 199, 1, 12, 0, 27})
459640430
```

3. Реализовать функцию для декодирования данных, содержащих битовые поля. В решении необходимо использовать побитовые операции.

Входные данные:

Шестнадцатиричная строка.

Выходные данные:

Список из битовых полей в виде пар имя-значение. Значения битовых полей имеют тип: десятичная строка.

Тесты:

```
>>> main('0x6a9f6')
[('R1', '6'), ('R2', '159'), ('R3', '5'), ('R4', '6'), ('R5', '0')]

>>> main('0x246d41')
[('R1', '1'), ('R2', '77'), ('R3', '3'), ('R4', '4'), ('R5', '4')]

>>> main('0x1bb448b')
[('R1', '11'), ('R2', '72'), ('R3', '5'), ('R4', '3'), ('R5', '49')]

>>> main('0xb077e5')
[('R1', '5'), ('R2', '382'), ('R3', '3'), ('R4', '0'), ('R5', '22')]
```

4. Реализовать функцию для декодирования данных, содержащих битовые поля. В решении необходимо использовать побитовые операции.

Входные данные:

Шестнадцатиричная строка.

Выходные данные:

Словарь из битовых полей. Значения битовых полей имеют тип: целое.

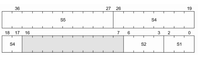
Тесты:

```
>>> main('0xb9a562')
{'N1': 34, 'N2': 21, 'N3': 26, 'N4': 3, 'N5': 5}

>>> main('0x85c1a5')
{'N1': 37, 'N2': 6, 'N3': 92, 'N4': 0, 'N5': 4}

>>> main('0xe3fb96')
{'N1': 22, 'N2': 46, 'N3': 63, 'N4': 0, 'N5': 7}

>>> main('0x2909e9')
{'N1': 41, 'N2': 39, 'N3': 16, 'N4': 1, 'N5': 1}
```

5. Реализовать функцию для транскодирования данных, содержащих битовые поля. В решении необходимо использовать побитовые операции. Неиспользуемые поля результата должны содержать нулевые биты.

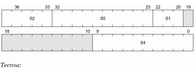
Входные данные:

Десятичная строка.



Выходные данные:

Шестнадцатиричная строка.



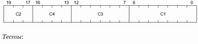
Тесты:

```
>>> main('239948252')
'0x1600c00376'

>>> main('128390864606')
'0x1fde602258'

>>> main('5170051890')
'0x1cc9e000cb'

>>> main('946680119137')
'0x1b89900113'
```

6. Реализовать функцию для транскодирования данных, содержащих битовые поля. В решении необходимо использовать побитовые операции.

Входные данные:

Десятичная строка.

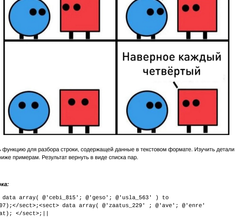


Выходные данные:

Десятичная строка.

Тесты:

```
>>> main('743096')
'748216'

>>> main('42860')
'791788'

>>> main('742832')
'486064'

>>> main('992703')
'517311'
```



7. Реализовать функцию для разбора строки, содержащей данные в текстовом формате. Изучить детали формата по приведенным ниже примерам. Результат вернуть в виде списка пар.

Пример 1

Входная строка:

```
    |<sect> data array( @'cebi_815'; @'geso'; @'usla_563' ) to
    q(dite_207);</sect><sect> data array( @'zaatus_229' ; @'ave'; @'enre'
    )to q(raat); </sect>;||
```

Разобранный результат:

```
    [('dite_207', ['cebi_815', 'geso', 'usla_563']), ('raat', ['zaatus_229', 'ave', 'enr
    e'])]
```

Пример 2

Входная строка:

```
    |<sect> data array( @'ated'; @'enso_554' ; @'edes_177') to
    q(lazaer);</sect>; <sect> data array( @'rela_539'; @'raradi' ;
    @'maques_294') to q(rive); </sect>; <sect> data array( @'vege_713'
    ;@'dier_208' ; @'eraxela' ) to q(erqu_432); </sect>; ||
```

Разобранный результат:

```
    [('lazaer', ['ated', 'enso_554', 'edes_177']),
    ('rive', ['rela_539', 'raradi', 'maques_294']),
    ('erqu_432', ['vege_713', 'dier_208', 'eraxela'])]
```

8. Реализовать функцию для разбора строки, содержащей данные в текстовом формате. Изучить детали формата по приведенным ниже примерам. Результат вернуть в виде словаря.

Пример 1

Входная строка:

```
    [ <data> set @"arerre_209" =-4948 </data><data>set @"zaen" = -5816
    </data> ]
```

Разобранный результат:

```
    {'arerre_209': -4948, 'zaen': -5816}
```

Пример 2

Входная строка:

```
    [<data>set @"este_583" = 5840 </data> <data> set @"usbear" = -2370
    </data> <data>set @"arso_10" = -1480 </data><data>set @"vesobi"= 2526
    </data> ]
```

Разобранный результат:

```
    {'este_583': 5840, 'usbear': -2370, 'arso_10': -1480, 'vesobi': 2526}
```

Выполнение данных заданий в течение практического занятия №4 даёт возможность получить следующие баллы:

- ***Любые 2 задачи: 0.5 балла***
- ***Любые 4 задачи: 1 балл***
- ***Любые 6 задач: 1.5 балла***
- ***Все 8 задач: 2 балла***