Рациональные числа - задача

- Сделать набор функций для работы с простыми дробями.
- Дробь хранится в виде [r_type, numer, denom]
- r_type - тип дроби: "basic" или "auto_simpl"
- При операциях с "auto_simpl" в отличии от "basic" нужно сокращать числитель и знаменатель на НОД
- Сделать поддержку функций add, sub, mul, tostr

Алгоритм евклида,

$$a \geq b, MCD(a,b) = \begin{cases} b, a \div b = 0 \\ MCD(b, a \div b) \end{cases}$$

Рациональные числа - API

```
1       r1 = ['basic', 1, 3]
2       r2 = ['basic', 1, 2]
3
4       print tostr(r1) # 1/3
5
6       r3 = sub(r2, r1)
7       print tostr(r3) # 1/6
8
9       r4 = add(r1, r1)
10      print tostr(r4) # 2/6
11
12      r5 = ['auto_simpl', r1[1], r1[2]]
13      r6 = add(r5, r5)
14      print tostr(r6) # 1/3
```

Рациональные числа - процедурный стиль

```
1       def tostr(x):
2           return "{[1]}/{[2]}".format(x)
3
4       def nod(x, y):
5           x = abs(x)
6           y = abs(y)
7           return _nod(max(x, y), min(x, y))
8
9       def _nod(x, y):
10          if y == 0:
11              return x
12          return nod(y, x % y)
13
14      def add(x, y):
15          nd = x[2] * y[2]
16          nn = x[1] * y[2] + x[2] * y[1]
17          if x[0] == 'basic':
18              return ['basic', nn, nd]
```

```
19            cur_nod = nod(nn, nd)
20            return ['auto_simpl', nn / cur_nod, nd / cur_nod]
21
22        def sub(x, y):
23            return add(x, [y[0], -y[1], y[2]])
```

Процедурный стиль - анализ

- Добавление новых типов требует изменения функции add
- `if x[0] == 'basic':` - ужасно
- Декомпозиция логики затруднена
- Перегрузка функций решает часть проблем, но только часть
- Один из вариантов решения - привязать функции к данным

Рациональные числа - не совсем процедурный стиль

```
1       def add_basic(x, y):
2           nd = x['denom'] * y['denom']
3           nn = x['num'] * y['denom'] + x['denom'] * y['num']
4           res = x.copy()
5           res['num'] = nn
6           res['denom'] = nd
7           return res
8       def add_simplified(x, y):
9           res = add_basic(x, y)
10          cur_nod = nod(res['num'], res['denom'])
11          res['num'] /= cur_nod
12          res['denom'] /= cur_nod
13          return res
14
15      x1 = {'num':1, 'denom':2, 'add': add_basic}
16
17      def add(x, y):
18          return x['add'](x, y)
```

Не совсем процедурный стиль - анализ

- Кода стало больше
- Его расширение упростилось - не нужно модифицировать функцию add, при добавлении нового типа
- Типовые теги стали менее нужны - тип это операции, которые есть у него
- Вместо `func(x)` теперь `x['func'](x)`. Для упрощения вызова старая процедурная семантика оставлена, но внутри нее перенаправление на новый вызов
- Однако если нужно написать новую функцию для всех типов, то все равно приходится использовать if/elif/elif/else. Только теги нужно вернуть
- Каждый экземпляр содержит большое количество ссылок на одни и те же функции
- Решение - вынесение всех методов в отдельный словарь, который все переменные данного типа используют совместно. Одновременно этот словарь становится типовым тегом

Рациональные числа - совсем не процедурный стиль

```
1   RationalNumber = {'add': add_basic,
2                     'sub': sub_basic}
3
4   RationalNumberSimpl = {'add': add_simplified,
5                          'sub': sub_simplified}
6
7   x1 = {'num':1, 'denom':2,
8         '__class__': RationalNumber}
9
10  x2 = {'num':1, 'denom':2,
11        '__class__': RationalNumberSimpl}
12
13  def add(x, y):
14      return x['__class__']['add'](x, y)
```

- Шаблон, использованный в функции add часто используется в python и позволяет имитировать перегрузку функций
- Именно так и устроено ООП в питоне внутри

Рациональные числа - классы

```
1   class BasicRational(object):
2       "basic rational number"
3
4       def __init__(self, num, denom):
5           self.num = num
6           self.denom = denom
7
8       def add(self, y):
9           nd = self.denom * y.denom
10          nn = self.num * y.denom + y.num * self.denom
11          return BasicRational(nn, nd)
12
13      def neg(self):
14          return BasicRational(-self.num, self.denom)
15
16      def sub(self, y):
17          return self.add(y.neg())
18
```

```
19          def tostr(self):
20              return "{}/{}".format(self.num, self.denom)
21
22      class AutoSimpl(BasicRational):
23          "Auto simplified rational number"
24
25          def add(self, y):
26              res = BasicRational.add(self, y)
27              cur_nod = nod(res.num, res.denom)
28              res.num /= cur_nod
29              res.denom /= cur_nod
30              return res
```

Рациональные числа - классы

```
1       class AutoSimpl(BasicRational):
2           "Auto simplified rational number"
3
4           def __init__(self, num, denom):
5               cur_nod = nod(num, denom)
6               self.num = num / cur_nod
7               self.denom = denom / cur_nod
```

Рациональные числа - интерфейсы питона

```
1   class BasicRational(object):
2       "basic rational number"
3
4       def __init__(self, num, denom):
5           self.num = num
6           self.denom = denom
7
8       def __add__(self, y):
9           nd = x[2] * y[2]
10          nn = x[1] * y[2] + x[2] * y[1]
11          return self.__class__(nn, nd)
12
13      def __neg__(self):
14          return self.__class__(-nn, nd)
15
16      def __sub__(self, y):
17          return self.add(y.neg())
18
```

```
19          def __str__(self):
20              return "{[1]}/{[2]}".format(x)
21
22          def __repr__(self):
23              return str(self)
24
25      b1 = AutoSimpl(1, 2)
26      b2 = AutoSimpl(1, 3)
27      b3 = b2 - b1 - b1
28      print b1, b2, b3
```

ООП vs Процедурный стиль

- (-) Часто больше кода
- (-) Усложняет язык
- (+) Уменьшает пересечение имен
- (+) Код лучше структурирован
- (+) Избавляет от ручной проверки типов
- (+) Упрощается расширение
- (+) Более высокий уровень абстракции упрощает построение программы путем выделения стандартных шаблонов проектирования
- (+) Многие из идей ООП имеют прямую поддержку в языке

Python ООП vs Процедурный стиль

- Возможность перегрузки функций
- Возможность перегрузки операторов