Студент (име и презиме):
Број индекса:
21. децембар 2019.

Факултет инжењерских наука
Основи програмирања
Други колоквијум

---

**Задаци:**

**Задатак 1** – [15% за решење + 5% за сложеност]
Написати процедуру за матрично множење матрица $\mathbf{X}$, димензија $n \times m$, и $\mathbf{Y}$, димензија $m \times p$, која као резултат даје матрицу $\mathbf{XY}$ димензија $n \times p$. Матрице су задате као дводимензионални, односно угнежђени низови низова бројева. Наиме $i$-ти елемент низа заправо је такође низ који представља $i$-ту врсту матрице, док његови чланови јесу бројеви који одговарају елементима дате колоне матрице.

решење

```
def множењеМатрица(X, Y):
    """X и Y: матрице целих (int), реалних (float), или комплексних (complex) бројева,
              представљене као угнежђени, то јест дводимензионални низови (низ низова)
              при чему важи да је број колона матрице X једнак броју врста матрице Y.
       враћа: нову матрицу (угнежђени низ) која представља матрични производ X * Y."""

    assert isinstance(X, list) and all([isinstance(X[i],list) for i in range(len(X))])
    assert isinstance(Y, list) and all([isinstance(Y[i],list) for i in range(len(Y))])
    assert len(set(map(len, X))) == 1 and len(set(map(len, Y))) == 1
    assert all([len(X[i]) == len(Y) for i in range(len(X))])

    резултат = [[0] * len(Y[0]) for _ in range(len(X))]
    for i in range(len(X)):
        for j in range(len(Y[0])):
            for k in range(len(Y)): # или range(len(X[0])):
                резултат[i][j] += X[i][k] * Y[k][j]
    return резултат

    # или решење у само једном реду:

    return [[sum(x * y for (x, y) in zip(ред, кол)) for кол in zip(*Y)] for ред in X]


```

решење

Асимптотска временска сложеност процедуре `множењеМатрица` је: $\mathcal{O}(nmp)$ то јест $\mathcal{O}(n^3)$ за $n = m = p$.

**Пример:** Позив `множењеМатрица([[1, 2], [3, 4], [5, 6], [7, 8]], [[1, 2, 3], [4, 5, 6]])` враћа као резултат следећу матрицу `[[9, 12, 15], [19, 26, 33], [29, 40, 51], [39, 54, 69]]`, зато што је $1 \cdot 1 + 2 \cdot 4 = 9$, затим $1 \cdot 2 + 2 \cdot 5 = 12$, и тако даље све до последњег члана $7 \cdot 3 + 8 \cdot 6 = 69$.

**Задатак 2** – [25%]
Написати класу `Разломак` која ће служити за представљање рационалних бројева као и за основне операције над њима. Скуп свих рационалних бројева $\mathbb{Q}$ се једноставно математички дефинише као:

$$\mathbb{Q} = \left\{ \frac{p}{q} : p \in \mathbb{Z}, q \in \mathbb{Z}, q \neq 0 \right\} \quad \text{, где је } \mathbb{Z} \text{ скуп свих целих бројева.}$$

Неопходно је подржати и основне операције као што су сабирање, одузимање, множење и дељење разломака, као и степеновање целим бројем, односно поређење два разломка и тражење максимума. Иако могу бити задати другачије, разломци се увек памте и исписују у свом најједноставнијем облику, то јест када су бројилац $p$ и именилац $q$ узајамно прости (када немају заједничких делитеља).

**Пример:** Позив `x, y, z = Разломак(1, 3), Разломак(5, 7), Разломак(3, 2)` након кога следи:

- наредба `print(x - Разломак(2) * y + z)`, би требало да штампа ниску `17/42`;
- наредба `print(y + y)`, би требало да штампа ниску `10/7`, а не несведен облик `70/49`;
- наредба `print(-x ** 2 + Разломак(19, 9))`, би требало само да штампа ниску `2`, а не `2/1`;
- наредба `print(+x ** 3 + z ** -3 - x)`, би требало да штампа само ниску `0`, а не `0/1` и слично;
- наредбе `print(x < y)` и `print(y < z)` ће штампати `True`, док ће `print(z < x)` штампти `False`;
- наредба `print(Разломак.максимум(x, y))` штампа исто што и `print(x.максимум(y))` тј. `5/7`;
- и коначно наредба `Разломак(1, 0)` подиже `ZeroDivisionError` изузетак због нуле у имениоцу док `x ** 0.5` подиже `AssertionError` уз напомену да експонент мора бити целобројна вредност.

решење

```
class Разломак(object):
      def __init__(self, бројилац, именилац = 1):
          if именилац == 0:
              raise ZeroDivisionError('Разломак(%s, 0)' % бројилац)
          def нзд(a, b):
              if b == 0: return a
              else:      return нзд(b, a % b)
          self.p = бројилац // нзд(бројилац, именилац)
          self.q = именилац // нзд(бројилац, именилац)

      def __pos__(self):
          return self

      def __neg__(self):
          return Разломак(-self.p, self.q)

      def __add__(self, други):
          return Разломак(self.p * други.q + други.p * self.q, self.q * други.q)


      def __sub__(self, други):
          return self.__add__(други.__neg__())


      def __mul__(self, други):
          return Разломак(self.p * други.p, self.q * други.q)


      def __truediv__(self, други):
          return Разломак(self.p * други.q, self.q * други.p)


      def __pow__(self, експонент):
          assert isinstance(експонент, int), 'експонент мора бити целобројна вредност'
          if експонент >= 0:
              return Разломак(self.p ** експонент, self.q ** експонент)
          else:
              return Разломак(self.q ** -експонент, self.p ** -експонент)

      def __lt__(self, други):
          return self.p * други.q < self.q * други.p


      def максимум(self, други):
          if self < други: return други
          else:            return self

      def __str__(self):
          if self.q == 1:
              return str(self.p)
          else:
              return str(self.p) + '/' + str(self.q)

```

решење

**Задатак 3 (SymPy)**
Претпоставити да је Пајтонова библиотека SymPy 1.4 учитана извршавањем следећег низа наредби:

```
   >>> from sympy import *
   >>> a, b, c, g, t, x, y = symbols('a b c g t x y')
```

**Задатак 3а** – [5%]
Вијетове формуле за полином другог степена, односно квадратну једначину $ax^2 + bx + c = 0$ гласе:

$$x_1 + x_2 = -\frac{b}{a} \quad \text{и} \quad x_1 x_2 = \frac{c}{a} \quad ,$$

где $x_1$ и $x_2$ представљају корене, то јест, решења дате једначине. Користећи се Вијетовим формулама, написати процедуру која враћа поворку збира и производа корена задатог полинома другог степена.

решење

```
def ВијетовеФормуле2(израз, x):
    """Враћа поворку збира и производа корена квадратног полинома задатог изразом."""
    return (-израз.coeff(x, 1)/израз.coeff(x, 2), израз.coeff(x, 0)/израз.coeff(x, 2))

```

решење

**Пример:** Позив `ВијетовеФормуле2(a*x**2 + b*x + c, x)` враћа поворку `(-b/a, c/a)` као резултат.

**Задатак 3б** – [5%]
Тангента функције јесте права која у датој тачки додирује криву и има исти нагиб као и функција. Једначина праве која представља тангенту криве задате функцијом $f(x)$ у тачки $x = x_0$ дата је као:

$$y = f(x_0) + f'(x_0)(x - x_0) \quad .$$

Написати процедуру која враћа једначину тангенте у тачки функције задате симболичким изразом.

решење

```
def тангента(израз, x, x0):
    """Враћа једначину тангенте функције променљиве x задате изразом у тачки x=x0."""
    return израз.subs(x, x0) + diff(израз, x).subs(x, x0) * (x - x0)

```

решење

**Примери:**

- Позив процедуре `тангента(x**2/2, x, 3)` ће вратити израз `3*x - 9/2` као једначину тангенте.
- Позив процедуре `тангента(x*sin(x**2) + 1, x, sqrt(pi))` враћа `-2*pi*(x - sqrt(pi)) + 1`.

**Задатак 3в** – [5%]
По дефиницији, функција $f(x)$ јесте непрекидна у тачки $x = x_0$ уколико важе следеће две једнакости:

$$\lim_{x \to x_0^-} f(x) = \lim_{x \to x_0^+} f(x) = f(x_0)$$

Користећи је, написати процедуру за проверу непрекидности функције задате симболичким изразом.

решење

```
def непрекидност(израз, x, x0):
    """Испитује да ли је функција задата изразом непрекидна у тачки x0 или није."""
    return limit(израз, x, x0, '-') == limit(израз, x, x0, '+') == израз.subs(x, x0)

```

решење

**Примери:**

- Позив процедуре `непрекидност(exp(1/x), x, 0)` враћа логичку вредност `False` као резултат.
- Позив процедуре `непрекидност(sin(x)/x, x, pi)` враћа логичку вредност `True` као резултат.

**Задатак 3г** – [5%]
Тренутну брзину тела $v(t)$ у било ком временском тренутку $t$ могуће је израчунати помоћу интеграла:

$$v(t) = v_0 + \int_0^t a(t)\, dt \quad ,$$

где $v_0$ представља почетну брзину у $t = 0$, а $a(t)$ произвољну функцију убрзања тела у времену. Написати процедуру која враћа тренутну брзину тела, ако је убрзање задато симболичким изразом.

решење

```
def тренутнаБрзина(израз, t, v0):
    """Враћа тренутну брзину, ако је убрзање задато изразом, а почетна брзина v0."""
    return v0 + integrate(израз, (t, 0, t))

```

решење

**Примери:**

- Позив процедуре `тренутнаБрзина(g, t, -10)` враћа `g*t - 10` као резултат ("слободни пад").
- Позив процедуре `тренутнаБрзина(sin(t), t, 0)` би требало да враћа `-cos(t) + 1` као резултат.

**Задатак 3д** – [5%]
Написати процедуру која израчунава вредност тренутне брзине у дискретним временским тренуцима. Наиме тренутна брзина се задаје преко симболичког израза у функцији времена $t$. Процедура такође прима као параметар и број тачака $n$ у којима је неопходно одредити тренутну брзину, а брзина бива срачуната на сваки секунд, односно у тренуцима 0 s, 1 s, 2 s, 3 s и тако даље све до $n$ s до броја тачака.

решење

```
def тренутнеБрзине(израз, t, n):
    """Враћа низ вредности тренутних брзина у свакој секунди од 0- до n-те секунде."""
    низТачака = []
    for тренутак in range(n + 1):
        низТачака.append(израз.subs(t, тренутак))
    return низТачака

```

решење

Позив `тренутнеБрзине(тренутнаБрзина(5,t,15), t, 7)` враћа `[15, 20, 25, 30, 35, 40, 45, 50]`.

**Задатак 4** – [15%]
Написати процедуру која проверава да ли се улазна ниска може претворити у реалан број тј. `float`.

решење

```
def даЛиЈеРеаланБрој(реаланБрој):
    """
    реаланБрој: произвољна непразна ниска која може бити исправно записан реаланБрој;
    Враћа логичку вредност тачно уколико је ниска исправан реалан број, иначе нетачно.
    """
    assert isinstance(реаланБрој, str) and len(реаланБрој) > 0
    регуларниИзраз = '\A\s*[+-]?(\d+(\.\d*)?|\.\d+)([eE][-+]?\d+)?\s*\Z'
    регуларниИзраз = '\A\s*[+-]?([0-9]+(\.[0-9]*)?|\.[0-9]+)([eE][-+]?[0-9]+)?\s*\Z'
    import re                                          |  try:
    образац = re.compile(регуларниИзраз, re.ASCII)     |     float(реаланБрој)
    if re.fullmatch(образац, реаланБрој):              |  except ValueError:
        return True                                    |     return False
    else:                                              |  else:
        return False                                   |     return True

```

решење

**Пример:** Позиви процедуре `даЛиЈеРеаланБрој` над аргументима `'123'`, `'-123.'`, `'.456'`, `'-123.456'`, `'123.456e-78'`, `'-123.456E78'` враћају `True`, док аргументи `'.'`, `'12.34.56'`, `'1e3.4'` враћају `False`.

**Задатак 5** – [15%]
Написати генератор позитивних рационалних бројева (разломака) који неким редоследом пролази кроз све (и сведене и несведене) позитивне рационалне бројеве облика $p/q$ где $p, q \in \mathbb{N}$, без понављања.

решење

```
def разломци():
    """
    Враћа све позитивне рационалне бројеве (разломке) као поворку бројиоца и имениоца.
    """
    збир = 1
    while True:
        збир += 1
        if збир % 2 == 0:
            бројилац, именилац = 1, збир - 1
            while именилац > 0: # или бројилац < збир:
                yield бројилац, именилац
                бројилац += 1
                именилац -= 1
        else:
            бројилац, именилац = збир - 1, 1
            while бројилац > 0: # или именилац < збир:
                yield бројилац, именилац
                бројилац -= 1
                именилац += 1

```

решење