$Αλγεβρα$
Μαθηματικά Γυμνασίου με Python

Δημήτρης Νικολός

21 Μαΐου 2020

Κεφάλαιο 1

Δεκαδικοί αριθμοί

1.1 Εισαγωγή

Αν χωρίσουμε τη μονάδα σε 10 ίσα μέρη τότε μπορούμε να πάρουμε κλάσματα της μονάδας όπως $\frac{3}{10}$, $\frac{5}{10}$ κλπ. Τα κλάσματα είναι ομώνυμα συγκρίνονται εύκολα και βοηθάνε στις πράξεις. Γενικότερα, ονομάζουμε δεκαδικό κλάσμα οποιδήποτε κλάσμα έχει παρονομαστή μια δύναμη του 10. Κάθε δεκαδικό κλάσμα γράφεται σαν δεκαδικός αριθμός με τόσα δεκαδικά ψηφία όσα μηδενικά έχει ο παρονομαστής του. Η Python χειρίζεται τους δεκαδικούς αριθμούς όπως και τους υπόλοιπους. Δοκίμασε:

```
>>> 0.3 + 0.5
0.8
>>> type(0.7)
<class 'float'>
```

Βλέπουμε ότι οι δεκαδικοί αριθμοί δεν είναι int, όπως οι ακέραιο αλλά float. Το όνομα float έχει να κάνει με τον τρόπο με τον οποίο ο υπολογιστής αποθηκεύει αποδοτικά αυτούς τους αριθμούς.

Ας συνδυάσουμε τις γνώσεις από τα κλάσματα με τα κλάσματα που μάθαμε στο προηγούμενο κεφάλαιο.

```
>>> from fractions import Fraction
>>> x = Fraction(3,10)
>>> float(x)
0.3
```

Το `Fraction(3,10)` εννοεί το κλάσμα $\frac{3}{10}$ που είναι ίσο με 0,3. Ομως στην Python το 0,3 θα το γράφουμε με 0.3. Με τη συνάρτηση float μετατρέπουμε το $\frac{3}{10}$ σε δεκαδικό αριθμό.

Ασκηση 1.1.1 *(Στο βιβλίο βρίσκεται στη Σελ. 56) Γράψτε τους αριθμούς* $\frac{3}{10}$, $\frac{825}{1000}$, $\frac{53}{1000}$, $\frac{1004}{10000}$.

```
>>> float(Fraction(3,10))
0.625
>>> float(Fraction(825,100))
8.25
>>> float(Fraction(53,1000))
0.053
>>> float(Fraction(1004,10000))
0.1004
```

Η Python μπορεί να μετατρέψει τα κλάσματα σε δεκαδικό αριθμό ανεξάρτητα από τον παρονομαστή.

Ασκηση 1.1.2 *(Στο βιβλίο βρίσκεται στη Σελ. 59) Γράψε καθένα από τα παρακάτω κλάσματα, ως δεκαδικό αριθμό: (i) με προσέγγιση εκατοστού και (ii) με προσέγγιση χιλιοστού:*

(α) $\frac{7}{16}$
(β) $\frac{21}{17}$
(γ) $\frac{20}{95}$

```
>>> x = Fraction(7,16)
>>> float(x)
0.4375
>>> round(float(x),2)
0.44
>>> round(float(x),3)
0.438
>>> x = Fraction(21,17)
>>> float(x)
1.2352941176470589
>>> round(float(x),2)
1.24
>>> round(float(x),3)
1.235
>>> x = Fraction(20,95)
>>> float(x)
0.21052631578947367
>>> round(float(x),2)
0.21
>>> round(float(x),3)
0.211
```

Η στρογγυλοποίηση των δεκαδικών υλοποιείται στην Python με τη συνάρτηση round. Οπότε μπορείς να στρογγυλοποιήσεις εύκολα δεκαδικούς αριθμούς ως εξής:

Ασκηση 1.1.3 *Να στρογγυλοποιήσεις τους παρακάτω δεκαδικούς αριθμούς στο δέκατο, εκατοστό και χιλιοστό:*

(α) 9876,008,
(β) 67,8956,
(γ) 0,001,
(δ) 8,239,
(ε) 23,7048.

Θυμόμαστε να αλλάζουμε την υποδιαστολή από κόμμα σε τελεία:

```
def roundall(x):
    print(round(x,1))
    print(round(x,2))
    print(round(x,3))

roundall(9876.008)
roundall(67.8956)
roundall(0.001)
roundall(8.239)
roundall(23.7048)
```

Το αποτέλεσμα είναι:

```
67.9
67.9
67.896
0.0
0.0
0.001
8.2
8.24
8.239
23.7
23.7
23.705
```

Ασκηση 1.1.4 *(Στο βιβλίο βρίσκεται στη Σελ. 59) Στον αριθμό* $34,\square\square\square$ *λείπουν τα τελευταία τρία ψηφία του. Να συμπληρώσεις τον αριθμό με τα ψηφία 9, 5 και 2, έτσι ώστε κάθε ψηφίο να γράφεται μία μόνο φορά. Να γράψεις όλους τους δεκαδικούς που μπορείς να βρεις και να τους διατάξεις σε φθίνουσα σειρά.*

Πώς μπορεί η Python να βρει όλους τους πιθανούς συνδυασμούς του 9,5,2; Δοκίμασε τη βιβλιοθήκη itertools και συγκεκριμένα τη συνάρτηση permutations.

```
>>> from itertools import permutations
>>> x = permutations([1,2,3])
>>> print(x)
<itertools.permutations object at 0x012BE1B0>
```

```
>>> print(list(x))
[(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)]
```

Ετσι με την permutations μπορείς να βρεις όλες τις αναδιατάξεις των αριθμών. Οπότε τώρα το πρόγραμμα μπορεί να γίνει ως εξής:

```
lista = []
from itertools import permutations
for p in permutations([9,5,2]):
    lista.append(34+p[0]/10+p[1]/100+p[2]/1000)
print(lista)
```

Που δίνει το αποτέλεσμα:

```
[34.952, 34.925000000000004, 34.592000000000006, 34.529,
34.29500000000001, 34.259]
```

Τα ψηφία που εμφανίζονται στο τέλος των αριθμών προκύπτουν από την αναπαράσταση των δεκαδικών στον υπολογιστή που υπόκειται σε κάποιους περιορισμούς. Αν δεν θέλουμε να εμφανίζονται μπορούμε να αλλάξουμε το for σε:

```
for p in permutations([9,5,2]):
    ar = 34+p[0]/10+p[1]/100+p[2]/1000
    lista.append(round(ar,3))
```

Τώρα για να γράψουμε τους αριθμούς με φθίνουσα σειρά θα δοκιμάσουμε τη sorted. Η sorted ταξινομεί τους αριθμούς που δίνονται σε μια λίστα. Δοκίμασε:

```
>>> sorted([4,2,3])
[2, 3, 4]
```

Ετσι το συνολικό πρόγραμμα γίνεται:

```
lista = []
from itertools import permutations
for p in permutations([9,5,2]):
    ar = 34+p[0]/10+p[1]/100+p[2]/1000
    lista.append(round(ar,3))
print(sorted(lista))
```

Που δίνει το αποτέλεσμα:

```
[34.259, 34.295, 34.529, 34.592, 34.925, 34.952]
```

Ομως η άσκηση μας ζητάει να τυπώσουμε τη λίστα με φθίνουσα σειρά. Αυτό μπορεί να γίνει δηλώνοντας στη sorted ότι θέλουμε αντίστροφη σειρά γράφοντας `reverse=True`. Το τελικό πρόγραμμα είναι το εξής:

```
lista = []
from itertools import permutations
for p in permutations([9,5,2]):
```

```
    ar = 34+p[0]/10+p[1]/100+p[2]/1000
    lista.append(round(ar,3))
print(sorted(lista,reverse=True))
```

Μια μικρή τροποποίηση που μπορεί να γίνει για να εμφανιστούν οι αριθμοί σε διαφορετικές γραμμές είναι να τυπώσουμε τη λίστα με μια for.

```
lista = []
from itertools import permutations
for p in permutations([9,5,2]):
    ar = 34+p[0]/10+p[1]/100+p[2]/1000
    lista.append(round(ar,3))

for x in sorted(lista,reverse=True):
    print(x)
```

Ασκηση 1.1.5 *(Στο βιβλίο βρίσκεται στη Σελ. 61) Να υπολογίσεις τα αθροίσματα:*
(α) $48, 18 + 3, 256 + 7, 129$
(β) $3, 59 + 7, 13 + 8, 195$

```
>>> 48.18+3.256+7.129
58.565
>>> 3.59 + 7.13 + 8.195
18.915
```

Ασκηση 1.1.6 *(Στο βιβλίο βρίσκεται στη Σελ. 61) Να υπολογίσεις το μήκος της περιμέτρου των οικοπέδων: (Σχήμα —)*

```
>>> 26.14 + 80.19 + 29.13+38.13+23.24+57.89+80.19
334.91
>>> 39.93+80.19+57.89+47.73+44.75+48.9+47.19
366.58
```

Ασκηση 1.1.7 *(Στο βιβλίο βρίσκεται στη Σελ. 61) Να κάνεις τις διαρέσεις: (α)* $579 : 48$
(β) $314 : 25$
(γ) $520 : 5, 14$
(δ) $49, 35 : 7$

```
>>> 579/48
12.0625
>>> 314/25
12.56
>>> 520/5.14
101.16731517509729
>>> 49.35/7
7.05
```

Ασκηση 1.1.8 *(Στο βιβλίο βρίσκεται στη Σελ. 61) Να κάνεις τις πράξεις:*

(α) $520 \cdot 0,1 + 0,32 \cdot 100$

(β) $4,91 \cdot 0,01 + 0,819 \cdot 10$

```
>>> 520*0.1 + 0.32*100
84.0
>>> 4.91*0.01 + 0.819*10
8.239099999999999
\end
```

Σε αυτή την άσκηση βλέπουμε ότι ο υπολογιστής προσεγγίζει τα αποτελέσματα με τον δικό του τρόπο. Δοκίμασε:

```
>>> x = 520*0.1 + 0.32*100
>>> x
84.0
>>> type(x)
<class 'float'>
>>> y = int(x)
>>> type(y)
<class 'int'>
>>> x == y
True
```

Αυτό σημαίνει πως ο ακέραιος αριθμός 84, και κάθε ακέραιος, στην Python μπορεί να αναπαρασταθεί σαν ακέραιος αλλά και σαν float με μηδενικά δεκαδικά ψηφία. Στην δεύτερη πράξη παρατηρούμε ότι αντί για το σωστό αποτέλεσμα που είναι $0,0491 + 8,19 = 8,2391$ η Python εμφανίζει μια προσέγγιση που είναι 8.239099999999999. Η διαφορά είναι πολύ μικρή. Ωστόσο οι δύο ποσότητες δεν είναι ίσες. Δοκίμασε:

```
>>> 4.91*0.01 + 0.819*10 == 8.2391
False
>>> 8.2391 - 4.91*0.01 + 0.819*10
1.7763568394002505e-15
```

Ο αριθμός `1.7763568394002505e-15` σημαίνει πως η διαφορά είναι περίπου 1.77 cot 10^{-15} που είναι πάρα πολύ μικρή και προκύπτει από τον τρόπο με τον οποίο η Python αποθηκεύει τους αριθμούς.

Ασκηση 1.1.9 *(Στο βιβλίο βρίσκεται στη Σελ. 61) Να κάνεις τις πράξεις:*

(α) $4,7 : 0,1 - 45 : 10$

(β) $0,98 : 0,0001 - 6785 : 1000$

```
>>> 4.7/0.1 - 45/10
42.5
>>> 0.98/0.0001 - 6785/1000
9793.215
```

Βλέπουμε ότι η Python υπολογίζει σωστά πρώτα τη διαίρεση και μετά την αφαίρεση.

Ασκηση 1.1.10 *(Στο βιβλίο βρίσκεται στη Σελ. 61) Η περίμετρος ενός τετραγώνου είναι 20,2. Να υπολογίσεις την πλευρά του.*

```
>>> 20.2/4
5.05
```

Ασκηση 1.1.11 *(Στο βιβλίο βρίσκεται στη Σελ. 61) Η περίμετρος ενός ισοσκελούς τριγώνου είναι 48,52. Αν η βάση του είναι 10,7, πόσο είναι η κάθε μία από τις ίσες πλευρές του;*

Αφαιρούμε πρώτα από το 48,52 το 10,7. Το αποτελέσμα το διαιρούμε με το δυο.

```
>>> 48.52-10.7
37.82000000000001
>>> 37.82/2
18.91
```

Ασκηση 1.1.12 *(Στο βιβλίο βρίσκεται στη Σελ. 61) Να υπολογίσεις τις τιμές των αριθμητικών παραστάσεων:*

(α) $24 \cdot 5 - 2 + 3 \cdot 5$

(β) $3 \cdot 11 - 2 + 45,1 : 2$

```
>>> 24*5 - 2 +3*5
133
>>> 3*11 - 2 + 54.1/2
58.05
```

Ασκηση 1.1.13 *(Στο βιβλίο βρίσκεται στη Σελ. 61) Να υπολογίσεις τις δυνάμεις:* *(α)* $3,1^2$, *(β)* $7,01^2$, *(γ)* $4,5^2$, *(δ)* $0,5^2$, *(ε)* $0,2^2$, *(στ)* $0,3^3$

```
>>> 3.1**2
9.610000000000001
>>> 7.01**2
49.1401
>>> 4.5**2
20.25
>>> 0.5**2
0.25
>>> 0.2**2
0.04000000000000001
>>> 0.3*3
0.8999999999999999
```

Πάλι κάνουν την εμφάνισή τους μικρές προσεγγίσεις.

Ασκηση 1.1.14 *Τοποθέτησε ένα "x" στην αντίστοιχη θέση (ΣΩΣΤΟ ΛΑΘΟΣ) (α)* $2,75+0,05+1,40+16,80=21$ *(β)* $420,510+72,490+45,19+11,81=500$ *(γ)* $4–3,852=1,148$ *(δ)* $32,01–4,001=28,01$ *(ε)* $41900\cdot 0,0001–0,0419\cdot 1000=0$ *(στ)* $56,89\cdot 0,01+4311:10000=1$ *(ζ)* $(3,2+7,2\cdot 2+24\cdot 0,1):100=0,2$

(α)

```
>>> 2.75 + 0.05 + 1.40 + 16.80 == 21
True
>>> 2.75 + 0.05 + 1.40 + 16.80
21.0
```

Αρα Σωστό

(β)

```
>>> 420.510 + 72.490 + 45.19 + 11.81 == 500
False
>>> 420.510 + 72.490 + 45.19 + 11.81
550.0
```

Αρα Λάθος

(γ)

```
>>> 4 - 3.852 == 1.148
False
>>> 4 - 3.852
0.14800000000000013
```

Αρα Λάθος

(δ)

```
>>> 32.01 - 4.001 == 28.01
False
>>> 32.01 - 4.001
28.008999999999997
```

Αρα Λάθος

(ε)

```
>>> 41900*0.0001 - 0.0419*1000 == 0
False
>>> 41900*0.0001 - 0.0419*1000
-37.71
```

Αρα Λάθος

(στ)

```
>>> 56.89*0.01 + 4311 / 10000 == 1
True
>>> 56.89*0.01 + 4311 / 10000
1.0
```

Αρα Σωστό

και

(ζ)

```
>>> (3.2 + 7.2*2 + 24*0.1) / 100 == 0.2
True
>>> (3.2 + 7.2*2 + 24*0.1) / 100
0.2
```

Αρα Σωστό.

1.2 Τυποποιημένη μορφή μεγάλων αριθμών

Η μάζα του Ηλιου είναι 1983000000000000000000000000000 κιλά. Για να διευκολυνθούμε μπορούμε να γράψουμε αυτόν τον αριθμό στην μορφή $\alpha \cdot 10$. Αρχικά μετράμε τα μηδενικά:

$$1983.000.000.000.000.000.000.000.000.000$$

ο αριθμός μας γίνεται:

$$1983 \cdot 10^{27}$$

Η τυποποιημένη μορφή απαιτεί το α να είναι μεγαλύτερο ή ίσο του 1 και μικρότερο του 10. Οπότε το 1983 πρέπει να γίνει 1,983 και ο αριθμός μας να γίνει

$$1,983 \cdot 10^{30}$$

σε τυποποιημένη μορφή μεγάλων αριθμών. Η Python υποστηρίζει την τυποποιημένη μορφή μεγάλων αριθμών με τη χρήση της εντολής format. Δοκίμασε:

```
>>> x = 1983000000000000000000000000000
>>> print(x)
1983000000000000000000000000000
>>> print('{:.3e}'.format(x))
1.983e+30
```

Η γενική μορφή είναι να χρησιμοποιείς <`'{:.Ne}'.format(x)`>όπου N είναι το πλήθος των δεκαδικών ψηφίων που θες να εμφανίζονται. Ετσι

```
>>> print('{:.2e}'.format(3140000000000000000))
3.14e+18
>>> print('{:.2e}'.format(234000000000000000))
2.34e+17
```

Η Python καταλαβαίνει την τυποποιημένη μορφή π.χ.:

```
>>> x = 3.14e+30
>>> print(x)
3.14e+30
>>> print('{:.0f}'.format(x))
3139999999999999741556248543232
```

Το ':.0f' σημαίνει πως ο αριθμός θα πρέπει να γραφεί σαν δεκαδικός (float) με μηδεν δεκαδικά ψηφία

Ασκηση 1.2.1 *(Στο βιβλίο βρίσκεται στη Σελ. 63) Να γράψεις τους παρακάτω αριθμούς στην τυποποιημένη μορφή: (α) 583.000 (β) 4.300.000 (γ) 7.960.000 (δ) 3.420.000.000 (ε) 4.800 (στ) 7.310 (ζ) 281.900 (η) 518.000.000 (θ) 131.000 (ι) 675.000.*

```
>>> print('{:.2e}'.format(583000))
5.83e+05
>>> print('{:.1e}'.format(4300000))
4.3e+06
>>> print('{:.2e}'.format(7960000))
7.96e+06
>>> print('{:.2e}'.format(3420000000))
3.42e+09
>>> print('{:.1e}'.format(4800))
4.8e+03
>>> print('{:.2e}'.format(7310))
7.31e+03
>>> print('{:.3e}'.format(281900))
2.819e+05
>>> print('{:.2e}'.format(518000000))
5.18e+08
>>> print('{:.2e}'.format(131000))
1.31e+05
>>> print('{:.2e}'.format(675000))
6.75e+05
```

Ασκηση 1.2.2 *(Στο βιβλίο βρίσκεται στη Σελ. 63) Να γράψεις τη δεκαδική μορφή των αριθμών: (α)* $3,1 \cdot 106$ *(β)* $4,820 \cdot 105$ *(γ)* $3,25 \cdot 104$ *(δ)* $7,4 \cdot 103$ *(ε)* $9,2 \cdot 102$*.*

```
>>> print(4.820 * 10**5)
482000.0
>>> print(3.25 * 10**4)
32500.0
>>> print(7.4 * 10**3)
7400.0
>>> print(9.2 * 10**2)
919.9999999999999
```

Ειδικά για το τελευταίο παρατηρούμε ότι έχουμε μια προσέγγιση και αντί για 920 που είναι το σωστό αποτέλεσμα προκύπτει 919.9999999999999. Η Python κάνει προσεγγίσεις όταν χρειάζεται να κάνει πράξεις ή όταν ο αριθμός δεν μπορεί να αναπαρασταθεί στα όρια του υπολογιστή. Το 920 δεν ανήκει στην δεύτερη κατηγορία οπότε το λάθος προκύπτει από την πράξη (πολλαπλασιαμός με το 100). Στην πραγματικότητα ο πολλαπλασιασμός αυτός δεν είναι αναγκαίος μπορείς να γράψεις 9.2e2 αντί για 9.2*10**2 και η Python θα καταλάβει τον αριθμό που θέλεις:

```
>>> 9.2e2
920.0
```