Μαθηματικά Γυμνασίου με Python

27 Σεπτεμβρίου 2021

Εισαγωγή

Το βιβλίο αυτό είναι ένας συνδυασμός των μαθηματικών που έμαθες στην Α΄ Γυμνασίου με τη γλώσσα προγραμματισμού Python. Θα θυμηθείς όσα έμαθες στην Α΄ Γυμνασίου και θα μάθεις και τα βασικά μιας σύγχρονης γλώσσας προγραμματισμού που χρησιμοποιείται από πολλούς προγραμματιστές σε όλον τον κόσμο.

Για να εγκαταστήσεις την Python στον υπολογιστή σου πήγαινε στη σελίδα https://www.python.org/ και κατέβασε την τελευταία έκδοση της Python 3 (Latest). Αφού κάνεις εγκατάσταση θα βρεις στον υπολογιστή σου το πρόγραμμα IDLE με το οποίο μπορείς να δουλέψεις αυτές τις σημειώσεις.

Κεφάλαιο 1

Φυσικοί αριθμοί

1.1 Οι αριθμοί και η Python

Οι φυσικοί αριθμοί είναι οι αριθμοί από 0, 1, 2, 3, 4, 5, 6, ..., 98, 99, 100, ..., 1999, 2000, 2001, ...

Η Python μπορεί να χειριστεί φυσικούς αριθμούς. Δοκιμάστε να γράψετε στο REPL έναν φυσικό αριθμό, θα δεις ότι η Python θα τον επαναλάβει. Π.χ. δες τον αριθμό εκατόν είκοσι τρια (123).

```
>>> 123
123
```

Στην Python όμως θα πρέπει να ακολουθείς κάποιους επιπλέον κανόνες. Για παράδειγμα στους αριθμούς δεν πρέπει να βάζεις τελείες στις χιλιάδες όπως στο χαρτί. Αν το κάνεις στην καλύτερη περίπτωση θα προκύψει κάποιο λάθος, στην χειρότερη ο υπολογιστής θα καταλάβει διαφορετικό αριθμό από αυτόν που εννοείς. Δες το παρακάτω παράδειγμα:

```
>>> 1.000.000
  File ”<stdin>”, line 1
    1.000.000
            ^
SyntaxError: invalid syntax
>>> 100.000
100.0
```

Σε αυτό το παράδειγμα, η Python δεν καταλαβαίνει καθόλου τον αριθμό 1.000.000 γραμμένο με τελείες ενώ μεταφράζει το 100.000 σε 100.0, που για την Python σημαίνει 100 (εκατό). Γι’ αυτόν τον λόγο δεν βάζουμε καθόλου τελείες έτσι αν θέλουμε να γράψουμε το ένα εκατομμύριο θα γράψουμε 1000000.

```
>>> 1000000
1000000
```

1.2 Πρόσθεση, αφαίρεση και πολλαπλασιασμός φυσικών αριθμών

Μια γλώσσα προγραμματισμού μπορεί να εκτελέσει απλές πράξεις πολύ εύκολα. Στο βιβλίο των μαθηματικών σου μπορείς να βρεις πολλές ασκήσεις με πράξεις. Μπορείς να τις λύσεις με την Python.

Άσκηση 1.2.1 *(Στο βιβλίο βρίσκεται στη Σελ. 16) Να υπολογιστούν τα γινόμενα:*

(α) $35 \cdot 10$*,*
(β) $421 \cdot 100$*,*
(γ) $5 \cdot 1.000$*,*
(δ) $27 \cdot 10.000$

Η python μπορεί να κάνει αυτές τις πράξεις ως εξής:

```
>>> 35*10
350
>>> 421*100
42100
>>> 5*1000
5000
>>> 27*10000
270000
```

Ο τελεστής του πολλαπλασιασμού είναι το αστεράκι * (SHIFT+8) στο πληκτρολόγιο. Εναλλακτικά, μπορείς να το βρείς στο αριθμητικό πληκτρολόγιο.

Άσκηση 1.2.2 *(Στο βιβλίο βρίσκεται στη Σελ. 16) Να εκτελεστούν οι ακόλουθες πράξεις:*

(α) $89 \cdot 7 + 89 \cdot 3$
(β) $23 \cdot 49 + 77 \cdot 49$
(γ) $76 \cdot 13 – 76 \cdot 3$
(δ) $284 \cdot 99$

```
>>> 89∗7+89∗3
890
>>> 23∗49+77∗49
4900
>>> 76∗13−76∗3
760
>>> 284∗99
28116
```

Στις παραπάνω περιπτώσεις η python εκτελεί πρώτα τους πολλαπλασιασμούς και μετά τις προσθέσεις/αφαιρέσεις δίνοντας έτσι το αποτέλεσμα που αναμένεται. Για παράδειγμα 897 + 893 = 623 + 267 = 890, που είναι το σωστό αποτέλεσμα.

Άσκηση 1.2.3 *(Στο βιβλίο βρίσκεται στη Σελ. 18) Υπολογίστε:*

(α) $157 + 33$
(β) $122 + 25 + 78$
(γ) $785 - 323$
(δ) $7.321 - 4.595$
(ε) $60 - (18 - 2)$
(στ) $52 - 11 - 9$
(ζ) $23 \cdot 10$
(η) $97 \cdot 100$
(ϑ) $879 \cdot 1.000$

Σε python τα παραπάνω υπολογίζονται ως εξής:

```
>>> 157+33
190
>>> 122+25+78
225
>>> 785−323
462
>>> 7321−4595
2726
>>> 60−(18−2)
44
>>> 52−11−9
32
>>> 23∗10
230
```

```
>>> 97*100
9700
>>> 879*1000
879000
```

Οι παρενθέσεις (SHIFT+9 και SHIFT+0) αλλάζουν τη σειρά των πράξεων. Οι πράξεις που είναι μέσα στην παρένθεση εκτελούνται πρώτες. Γι’ αυτό το λόγο 60-(18-2)=60-16=44.

Άσκηση 1.2.4 *(Στο βιβλίο βρίσκεται στη Σελ. 18) Σε ένα αρτοποιείο έφτιαξαν μία μέρα 120 κιλά άσπρο ψωμί, 135 κιλά χωριάτικο, 25 κιλά σικάλεως και 38 κιλά πολύσπορο. Πουλήθηκαν 107 κιλά άσπρο ψωμί, 112 κιλά χωριάτικο, 19 κιλά σικάλεως και 23 κιλά πολύσπορο. Πόσα κιλά ψωμί έμειναν απούλητα;*

Με τις γνώσεις που έχουμε θα πρέπει να μετατρέψουμε το παραπάνω πρόβλημα σε μια αριθμητική παράσταση ώστε η python να μπορεί να την υπολογίσει, στη συγκεκριμένη περίπτωση η σωστή παράσταση είναι:

$$(120-107)+(135-112)+(25-19)+(38-23)$$

```
>>> (120−107)+(135−112)+(25−19)+(38−23)
57
```

και η απάντηση είναι 57 κιλά ψωμί.

1.3 Δυνάμεις φυσικών αριθμών

Ο τελεστής της python για τις δυνάμεις είναι ο ** (δυο φορές το αστεράκι). Δηλαδή, αν θέλουμε να υπολογίσουμε το 10^2 θα γράψουμε 10**2, με όμοιο τρόπο μπορούμε να υπολογίσουμε και τις υπόλοιπες δυνάμεις. Δοκίμασε τα παρακάτω στο REPL.

```
>>> 10**2
100
>>> 10**3
1000
>>> 10**4
10000
>>> 10**5
100000
```

```
>>> 10**6
1000000
```

Στη προτεραιότητα των πράξεων, οι δυνάμεις έχουν μεγλύτερη προτεραιότητα από τον πολλαπλασιασμό και την πρόσθεση. Οπότε όταν έχουμε και δυνάμεις σε μια παράσταση πρώτα γίνονται οι πράξεις στις παρενθέσεις, μετά οι δυνάμεις και μετά οι πολλαπλασιασμοί και οι προσθέσεις. Την ίδια σειρά ακολουθεί και η python για τον υπολογισμό των πράξεων.

Άσκηση 1.3.1 *(Στο βιβλίο βρίσκεται στη Σελ. 21) Να εκτελεστούν οι πράξεις*

1. $(2 \cdot 5)^4 + 4 \cdot (3+2)^2$
2. $(2+3)^3 - 8 \cdot 3^2$

Οι αντίστοιχες εκφράσεις είναι (2*5)**4+4*(3+2)**2 και (2+3)**3 - 8*3**2.

```
>>> (2*5)**4+4*(3+2)**2
10100
>>> (2+3)**3 - 8*3**2
53
```

Η 8*3**2 υπολογίζεται ως $8 \cdot (3^2)$, δηλαδή $8 \cdot 9 = 72$, αφού πρώτα γίνεται η δύναμη και μετά οι πολλαπλασιασμοί.

Άσκηση 1.3.2 *Κάνε τις πράξεις: (α)* $3 \cdot 5^2$,

(β) $3 \cdot 5^2 + 2$,
(γ) $3 \cdot 5^2 + 2^2$,
(δ) $3 \cdot 5 + 2^2$,
(ε) $3 \cdot (5+2)^2$.

Αυτές οι πράξεις μπορούν να γίνουν στο REPL.

```
>>> 3*5**2
75
>>> 3*5**2 + 2
77
>>> 3*5**2 + 2**2
79
>>> 3*5 +2**2
19
>>> 3*(5 + 2)**2
147
```

Άσκηση 1.3.3 *Κάνε τις πράξεις: (α)* $3^2 + 3^3 + 2^3 + 2^4$*,*
(β) $(13 - 2)^4 + 5 \cdot 3^2$

```
>>> 3**2 +3**3 +2**3 +2**4
60
>>> (13-2)**4 + 5*3**2
14686
```

Άσκηση 1.3.4 *Βρες τις τιμές των παραστάσεων:*
(α) $(6 + 5)^2$ *και* $6^2 + 5^2$*,*
(β) $(3 + 6)^2$ *και* $3^2 + 6^2$*.*

```
>>> (6+5)**2
121
>>> 6**2+5**2
61
>>> (3+6)**2
81
>>> 3**2+6**2
45
```

1.4 Συγκρίσεις φυσικών αριθμών

Μπορούμε να συγκρίνουμε αριθμούς στην Python χρησιμοποιώντας τους τελεστές == (πληκτρολογούμε δύο φορές το =) για την *ισότητα*, > για το *μεγαλύτερο* και < για το *μικρότερο*. Επίσης μπορούμε να χρησιμοποιήσουμε >= για το *μεγαλύτερο ή ίσο* και <= για το *μικρότερο ή ίσο*, τέλος υπάρχει το != για το *δεν είναι ίσο*. Μπορείς να δοκιμάσεις τα παρακάτω:

```
>>> 123==123
True
>>> 123>123
False
>>> 123>122
True
>>> 123<123
False
>>> 123<124
```

```
True
>>> 123<=123
True
>>> 123<=124
True
>>> 123<=122
False
>>> 123>=123
True
>>> 123>=124
False
>>> 123>=122
True
>>> 122 != 123
True
>>> 122 != 122
False
```

Η Python επιστρέφει True (αληθές) όταν μία πρόταση ισχύει και False (ψευδές) όταν δεν ισχύει.

Σκέψου ότι για την Python η σύγκριση είναι και αυτή μια πράξη. Αντί η πράξη αυτή να δίνει σαν αποτέλεσμα έναν αριθμό δίνει σαν αποτέλεσμα το αληθές ή το ψευδές.

Για παράδειγμα:

Άσκηση 1.4.1 *Να συγκρίνετε τα* 3^2 *και* 2^3.

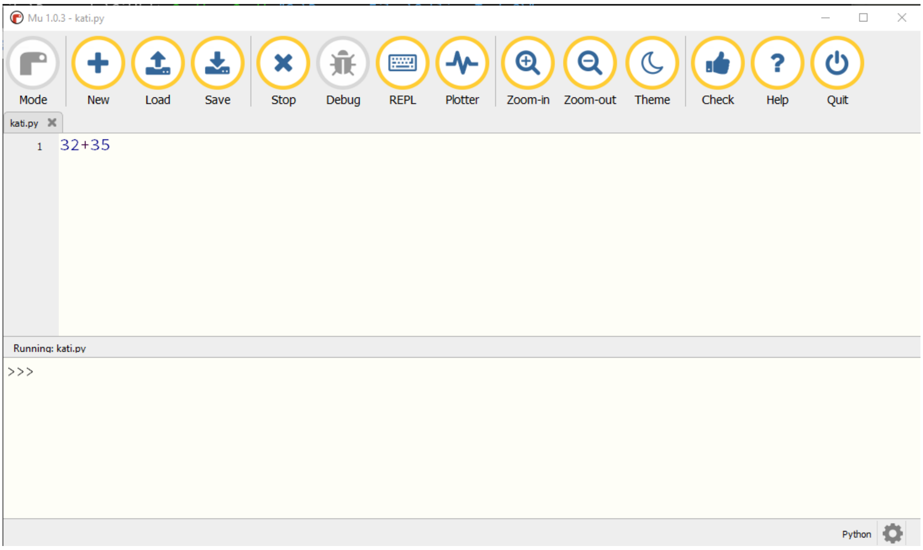
Η σύγκριση αυτή μπορεί να γίνει στο REPL. Δοκίμασε:

```
>>> 3**2 > 2**3
True
```

Άρα το 3^2 είναι μεγαλύτερο από το 2^3. Θυμήσου ότι το $3^2 = 9$, ενώ $2^3 = 8$.

1.5 Η εντολή print

Ήρθε η ώρα να γράψεις εντολές στο πάνω παράθυρο, δηλαδή να γράψεις το πρώτο σου πρόγραμμα. Με βάση όσα ξέρεις προσπάθησε να γράψεις μια πράξη στο πάνω παράθυρο, για παράδειγμα $32 + 35$. Ύστερα πάτησε το κουμπί της εκτέλεσης (Run). Μπορείς να δεις το αποτέλεσμα στην εικόνα 1.1.



Σχήμα 1.1: Η εκτέλεση δεν δίνει κάποιο αποτέλεσμα

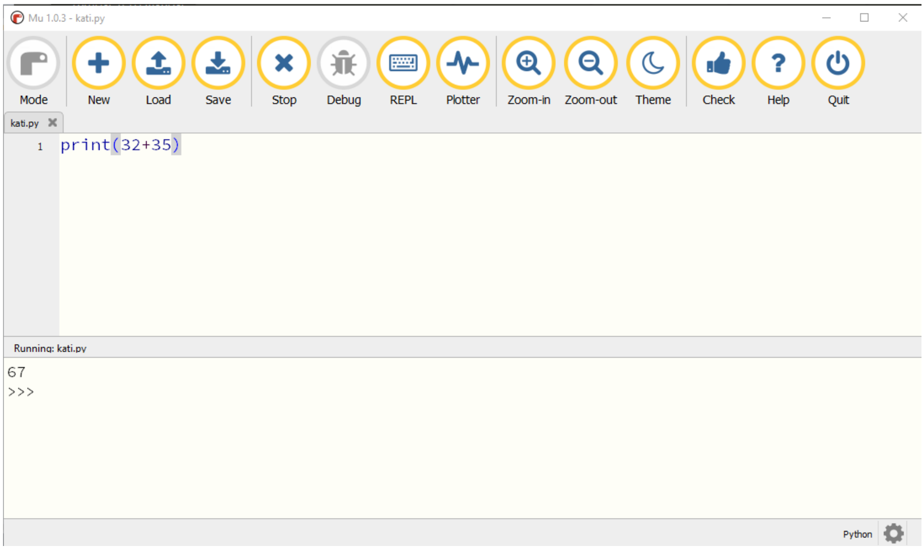
Η Python εκτελεί την πράξη $32+35$, και υπολογίζει το αποτέλεσμα. Αν δεν το έκανε και υπήρχε κάποιο πρόβλημα θα εμφάνιζε κάποιο μήνυμα λάθους στο REPL. Το υπολογισμένο αποτέλεσμα δεν εμφανίζεται. Για να εμφανιστεί το αποτέλεσμα πρέπει να χρησιμοποιήσεις την εντολή print (εκτύπωσε). Η εντολή print εκτελείται ως εξής:

```
print(32+35)
```

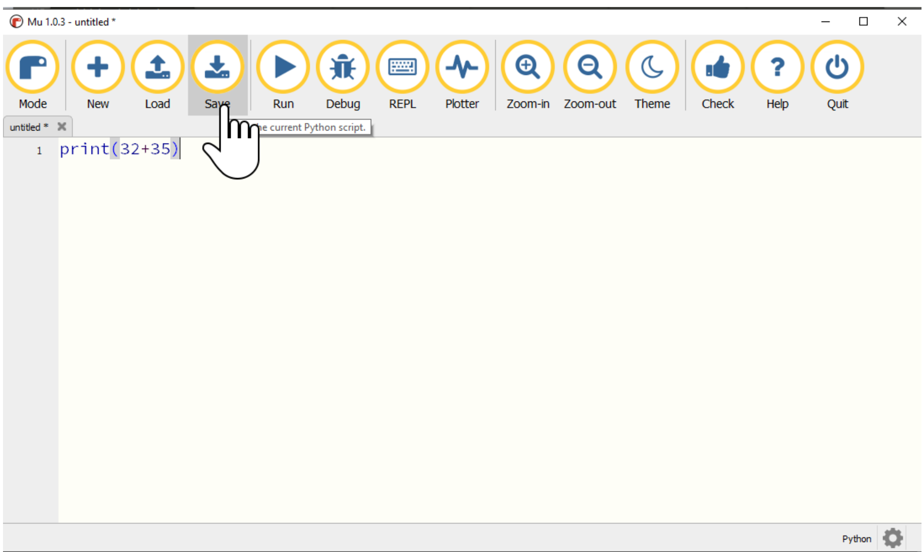
Γράφουμε δηλαδή, print ανοίγουμε παρένθεση, γράφουμε αυτό που θέλουμε να εκτυπωθεί και κλείνουμε την παρένθεση. Όταν εκτελέσουμε το πρόγραμμα με την print τότε εμφανίζεται το αποτέλεσμα στο REPL (εικόνα 1.2). Μόλις έγραψες το πρώτο σου πρόγραμμα στην Python. Μάλιστα το πρόγραμμά σου κάνει κάτι. Υπολογίζει το αποτέλεσμα της πράξης $32+35$. Μπορείς να αποθηκεύσεις το πρόγραμμά σου στον υπολογιστή σου κάνοντας κλικ στο εικονίδιο Save του Mu (εικόνα 1.3).

1.6 Απαρίθμηση

Είδαμε ότι η Python μπορεί να κάνει πολύ γρήγορα, πολύπλοκες πράξεις ακόμη και με δυνάμεις, αλλά δεν είδαμε ακόμη τις απλές ασκήσεις που υπάρχουν στις πρώτες σελίδες του βιβλίου. Όπως για παράδειγμα ποιοι είναι οι τρεις



Σχήμα 1.2: Η εκτέλεση δίνει το αποτέλεσμα της πράξης



Σχήμα 1.3: Αποθήκευση με το Mu

προηγούμενοι αριθμοί του 289 και ποιο οι δύο επόμενοι (Στο βιβλίο βρίσκεται στη Σελ. 13) .

Σε ένα πρόγραμμα χρειαζόμαστε δεδομένα εισόδου και εξόδου, μπορούμε να γράψουμε ένα πρόγραμμα που ννα δέχεται σαν είσοδο τον βασικό αριθμό και το πόσους αριθμούς πριν και μετά θα τυπώσουμε. Μέχρι να φτάσουμε εκεί μπορούμε να λύσουμε το πρόβλημα της σελίδας 13 με το παρακάτω πρόγραμμα:

```
print(289−3)
print(289−2)
print(289−1)
print(289+1)
print(289+2)
```

που δίνει το αποτέλεσμα

```
286
287
288
290
291
```

Πιο σωστό θα ήταν να τυπώσουμε κάποια μηνύματα ενδιάμεσα. Σε αυτή την περίπτωση θα γράψουμε τις παρακάτω εντολές.

```
print(” Οι προηγούμενοι αριθμοί είναι:”)
print(289−3)
print(289−2)
print(289−1)
print(” Οι επόμενοι αριθμοί είναι:”)
print(289+1)
print(289+2)
```

Για να εμφανίσει η print τις λέξεις που θέλουμε πρέπει να τις βάλουμε μέσα σε εισαγωγικά. Η Python υποστηρίζει είτε μονά εισαγωγικά, είτε διπλά. Αυτά εισάγονται συνήθως με το ίδιο κουμπί του πληκτρολογίου (κοντά στο ENTER), είτε με SHIFT ή χωρίς. Θυμήσου να κλείνεις τα εισαγωγικά με τον ίδιο τρόπο που τα άνοιξες. Στο πρόγραμμα Mu τα εισαγωγικά αυτά δεν φαίνονται όπως σε άλλα πρόγραμματα σαν ‘Εισαγωγικά’ ή “Εισαγωγικά ” ή «Εισαγωγικά», αλλά φαίνονται κάπως πιο απλά και ίδια στο άνοιγμα και το κλείσιμο ’Εισαγωγικά’ ή ”Εισαγωγικά”.

Αν θέλουμε να αλλάξουμε το 289 και να βάλουμε έναν άλλο αριθμό,π.χ. το 132 θα πρέπει να αντικαταστήσουμε το 289 μέσα σε όλες τις εντολές print με το 132.

```
print(”Οι προηγούμενοι αριθμοί είναι:”)
print(132−3)
print(132−2)
print(132−1)
print(”Οι επόμενοι αριθμοί είναι:”)
print(132+1)
print(132+2)
```

Υπάρχει όμως ένας καλύτερος τρόπος, ο τρόπος αυτός είναι να δώσουμε ένα όνομα στον αριθμό μας. Μπορούμε να πούμε ότι το n είναι το όνομα του αριθμού. Αυτό γίνεται με την εντολή n=132. Τότε το πρόγραμμά μας γίνεται:

```
n = 132
print(”Οι προηγούμενοι αριθμοί είναι:”)
print(n−3)
print(n−2)
print(n−1)
print(”Οι επόμενοι αριθμοί είναι:”)
print(n+1)
print(n+2)
```

Μετά την εντολή n=132 η Python ξέρει ότι το n είναι ένα όνομα για το 132 και μπορεί να κάνει πράξεις με αυτό. Για παράδειγμα n+1 κάνει τώρα 133.

Αν θέλουμε να κάνουμε τώρα το ίδιο πρόγραμμα αλλά όχι για το 132 αλλά για το 210, χρειάζεται να αλλάξουμε μόνο μία γραμμή και το πρόγραμμά μας να γίνει ως εξής:

```
n = 210
print(”Οι προηγούμενοι αριθμοί είναι:”)
print(n−3)
print(n−2)
print(n−1)
print(”Οι επόμενοι αριθμοί είναι:”)
print(n+1)
print(n+2)
```

Στην Python, όταν δίνουμε ένα όνομα σε έναν αριθμό (με τον τελεστή =) τότε δημιουργούμε μια μεταβλητή. Η μεταβλητή έχει ένα όνομα, στην περίπτωσή μας το n, και μια τιμή, στην περίπτωσή μας το 210.

Αν αντί για τους επόμενους δύο αριθμούς θέλαμε τους επόμενους **δέκα** θα γράφαμε ένα πρόγραμμα όπως το παρακάτω:

```
n = 210
print(n)
print(n+1)
print(n+2)
print(n+3)
print(n+4)
print(n+5)
print(n+6)
print(n+7)
print(n+8)
print(n+9)
print(n+10)
```

Το παραπάνω πρόγραμμα εμφανίζει και τον αριθμό μας n, δηλαδή το 210.

Για να μην γράφουμε πολλές εντολές όταν κάνουμε το ίδιο πράγμα χρησιμοποιούμε την εντολή for. Το πρόγραμμά μας με την for μπορεί να γίνει:

```
n = 210
for i in 0,1,2,3,4,5,6,7,8,9,10:
    print(n+i)
```

Όταν γράψεις την for στην Python θα πρέπει να δηλώσεις ποιες εντολές θα εκτελεστούν πολλές φορές. Αυτή η δήλωση γίνεται βάζοντας αυτές τις εντολές λίγο πιο μέσα χρησιμοποιώντας το πλήκτρο κενό ή το πλήκτρο tab. Μια καλή πρακτική είναι να βάζεις τέσσερα κενά. Έτσι, πριν την εντολή print(n+i) βάζεις τέσσερα κενά δηλαδή ⍰⍰⍰⍰print(n+i). Το πρόγραμμα αυτό σημαίνει πως για το i μέσα στο σύνολο 0, 1, 2, 3, ...10 και με αυτή τη σειρά εμφάνισε το n+i. Έτσι το αποτέλεσμα είναι το αναμενόμενο

```
210
211
212
213
214
215
216
217
218
219
```

```
220
```

Στην Python υπάρχει ένας πιο εύκολος τρόπος να γράψουμε τους αριθμούς από το 0 έως το 10. Αυτός ο τρόπος είναι η εντολή range και συγκεκριμένα η range(11). Η range(11) φτιάχνει τους αριθμούς από το 0 μέχρι το 10 οι οποίοι είναι σε πλήθος 11.

```
>>> list(range(11))
[0,1,2,3,4,5,6,7,8,9,10]
```

Έτσι το πρόγραμμά μας γίνεται:

```
n = 210
for i in range(11):
    print(n+i)
```

Οπότε μπορούμε να δώσουμε όνομα στο παραπάνω πρόγραμμα ως εξής:

```
def meta(n,d):
    for i in range(d+1):
        print(n+i)
```

Θα έχουμε το παραπάνω αποτέλεσμα αν εκτελέσουμε το πρόγραμμα ως εξής:

```
>>> meta(210,10)
```

Μπορούμε να γράψουμε το πρόγραμμα για τους αριθμούς πριν και μετά ως εξής:

```
def prinmeta(n,d):
  for i in range(d,0,-1):
    print(n-i)
  for i in range(d+1):
    print(n+i)
```

Αν γράψουμε την παρακάτω εντολή:

```
>>> prinmeta(210,10)
200
201
202
203
204
205
```

```
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
```

Με το πρόγραμμα αυτό μπορούμε να δοκιμάσουμε και το εξής:

```
>>> prinmeta(283,3)
280
281
282
283
284
285
286
```

1.7 Στρογγυλοποίηση

Το βιβλίο των Μαθηματικών της Α’ Γυμνασίου αναφέρει πως Για να στρογγυλοποιήσουμε έναν φυσικό αριθμό (Στο βιβλίο βρίσκεται στη Σελ. 12) :

1. Προσδιορίζουμε την τάξη στην οποία θα γίνει η στρογγυλοποίηση

2. Εξετάζουμε το ψηφίο της αμέσως μικρότερης τάξης

3. Αν αυτό το ψηφίο είναι μικρότερο του 5 (δηλαδή 0, 1, 2, 3 ή 4) το ψηφίο αυτό και όλα τα ψηφία των υπόλοιπων τάξεων μηδενίζονται.

4. Αν είναι μεγαλύτερο ή ίσο του 5 (δηλαδή 5, 6, 7, 8 ή 9) το ψηφίο αυτό και όλα τα ψηφία των υπόλοιπων τάξεων αντικαθίστανται από το 0 και το ψηφίο της τάξης στρογγυλοποίησης αυξάνεται κατά 1.

Ας πούμε ότι θέλουμε να στρογγυλοποιήσουμε τον αριθμό 454.018.512 στα εκατομμύρια. Η απάντηση που περιμένουμε είναι 454 εκατομμύρια. Για να τα καταφέρουμε θα χρησιμοποιήσουμε την διαίρεση. Όμως στην Python υπάρχουν *δύο* διαιρέσεις μία με το σύμβολο / και μία με το σύμβολο //. Ας δούμε τις διαφορές τους στο REPL.

```
>>> x = 454018512
>>> print(x/1000000)
454.018512
>>> print(x//1000000)
454
```

Η «κανονική» διαίρεση, με τη μία κάθετο /, δίνει το αποτέλεσμα της διαίρεσης με τα δεκαδικά ψηφία. Η «ακέραια» διαίρεση δίνει μόνο τον ακέραιο αριθμό. Δεν μπορούμε να πούμε ότι η ακέραια διαίρεση θα μας δώσει την στρογγυλοποίηση γιατί η ακέραια διαίρεση δεν στρογγυλοποιεί τα δεκαδικά ψηφία αλλά τα απορρίπτει εντελώς. Έτσι, ακόμη και αν είχαμε 454918512 κατοίκους η ακέραια διαίρεση θα δώσει 454 αντί για το στρογγυλοποιημένο που είναι 455.

```
>>> x = 454918512
>>> print(x/1000000)
454.918512
>>> print(x//1000000)
454
```

Χρειάζεται επομένως να δούμε το ψηφίο της αμέσως χαμηλότερης τάξης το οποίο είναι το πρώτο δεκαδικό της κανονικής διαίρεσης. Για να το απομονώσουμε αφαιρούμε από το αποτέλεσμα της κανονικής διαίρεσης το ακέραιο μέρος.

```
>>> x = 454018512
>>> x/1000000 - x//1000000
0.018511999999986983
```

Οπότε τώρα έχουμε δύο ενδεχόμενα αν το αποτέλεσμα αυτής της πράξης είναι μικρότερο από $0,5$, όπως παραπάνω, τότε το αποτέλεσμα που ψάχνουμε είναι το αποτέλεσμα της ακέραιας διαίρεσης. Αλλιώς πρέπει να προσθέσουμε τον αριθμό ένα στο αποτέλεσμα της ακέραιας διαίρεσης. Αυτό γίνεται με την εντολή if, που σημαίνει στα αγγλικά αν. Για ευκολία μπορούμε να ονομάσουμε d την διαφορά των δύο διαιρέσεων με την εντολή:

```
d = x/1000000 − x//1000000
```

Επειδή το πρόγραμμα γίνεται μεγαλύτερο τώρα θα το γράψουμε στο πάνω παράθυρο του Mu.

```
x = 454018512
d = x/1000000 − x//1000000
if d < 0.5:
    print(x//1000000)
else:
    print(x//1000000 + 1)
```

Την if την γράφουμε ως εξής:

```
if συνθήκη:
    εντολές που εκτελούνται
    αν ισχύει η συνθήκη
else:
    εντολές που εκτελούνται
    αν δεν ισχύει η συνθήκη
```

Θυμήσου να βάζεις την άνω κάτω τελεία (:) μετά τη συνθήκη και μετά τη λέξη else.

Αν στο ίδιο πρόγραμμα και βάλεις αντί για 454.018.512 τον αριθμό 454.918.512 θα δεις ότι θα εμφανιστεί το σωστό αποτέλεσμα (455).

Αν θέλεις στρογγυλοποίηση στις χιλιάδες τότε το πρόγραμμά σου γίνεται:

```
x = 454018512
d = x/1000 − x//1000
if d < 0.5:
    print(x//1000)
else:
    print(x//1000 + 1)
```

και το αποτέλεσμα είναι 454019.

Άσκηση 1.7.1 *Για να γίνει το 454.018.512, 450 εκατομμύρια (Στο βιβλίο βρίσκεται στη Σελ. 12) η στρογγυλοποίηση γίνεται στις δεκάδες των εκατομμυρίων. Μπορείς να γράψεις ένα πρόγραμμα που να στρογγυλοποιεί αριθμούς στις δεκάδες των εκατομμυρίων;*

Η απάντηση με βάση το παραπάνω πρόγραμμα είναι η εξής:

```
x = 454018512
d = x/10000000 − x//10000000
if d < 0.5:
    print(x//10000000*10)
else:
    print((x//10000000 + 1)*10)
```

Σε αυτή την περίπτωση από τον αριθμό 454.018.512,00 πρέπει να φτάσουμε πρώτα στις δεκάδες των εκατομμυρίων και μετά πολλαπλασίαζουμε με το δέκα ώστε να μην απαντήσει το πρόγραμμα 45 (δεκάδες εκατομμύρια) αλλά 450 (εκατομμύρια).

1.8 Επανάληψη στις πράξεις

Άσκηση 1.8.1 *Συμπλήρωσε τον πίνακα τα τετράγωνα και τους κύβους των αριθμών από το 8 μέχρι το 25 (Στο βιβλίο βρίσκεται στη Σελ. 22) .*

```
for a in range(8,26):
    print(a**2,end =" ")
print()
print()
for a in range(8,26):
    print(a**3,end=" ")
```

Το αποτέλεσμα αυτού του προγράμματος είναι:

```
64 81 100 121 144 169 196 225 256 289 324 361 400 441 484 529 576 625

512 729 1000 1331 1728 2197 2744 3375 4096 4913 5832 6859 8000 9261
10648 12167 13824 15625
```

Η εντολή print μπορεί να πάρει περισσότερα από ένα ορίσματα, το πρώτο όρισμα είναι αυτό που θα εμφανίσει. Το δεύτερο όρισμα που δώσαμε είναι το end και το ορίσαμε ίσο με το κενό (end="") που σημαίνει ότι η print όταν εμφανίσει το πρώτο όρισμα δεν θα αλλάξει γραμμή αλλά θα αφήσει ένα κενό. Η εντολή print() αλλάζει απλά γραμμή.

Άσκηση 1.8.2 *Βρες τα τετράγωνα των αριθμών 10,20,30,40,50,60,70,80 και 90 (Στο βιβλίο βρίσκεται στη Σελ. 22) .*

Το πρόγραμμα είναι το εξής:

```
for i in range(10,100,10):
    print(i**2,end=',')
```

και το αποτέλεσμα της εκτέλεσης του προγράμματος είναι

```
100,400,900,1600,2500,3600,4900,6400,8100,
```

Άσκηση 1.8.3 *Βρες τους κύβους των αριθμών 10,20,30,40,50*

```
for i in range(10,60,10):
    print(i**3,end=', ')
```

Το αποτέλεσμα της εκτέλεσης είναι:

```
1000, 8000, 27000, 64000, 125000,
```

1.9 Ανάπτυγμα

Άσκηση 1.9.1 *(Στο βιβλίο βρίσκεται στη Σελ. 21) Να γραφεί το ανάπτυγμα του αριθμού 7.604 με χρήση των δυνάμεων του 10.*

Η απάντηση είναι $7 \cdot 10^3 + 6 \cdot 10^2 + 0 \cdot 10^1 + 4$. Με συμβολισμό της Python η απάντηση που περιμένουμε είναι:

```
7*10**3+6*10**2+0*10+4
```

Ας υποθέσουμε ότι ξέρουμε ότι ο αριθμός είναι τετραψήφιος, πως μπορούμε να βρούμε το ανάπτυγμα του. Ξεκινάμε από το πρώτο ψηφίο. Ποιο είναι το πρώτο ψηφίο; Το πρώτο ψηφίο προκύπτει αν διαιρέσουμε τον αριθμό με το 1000 και κρατήσουμε το ακέραιο μέρος. Δοκίμασε στο REPL:

```
>>> 7604//1000
7
```

Βρήκες το πρώτο ψηφίο, πώς μπορείς να βρεις το δεύτερο; Ας διαιρέσουμε με το 100.

```
>>> 7604//100
76
```

Στην ουσία δεν μπορείς να διαιρέσεις τον αρχικό αριθμό με το 100 αλλά αυτόν που σου μένει αφού αφαιρέσεις το πρώτο ψηφίο που έχεις ήδη βρει δηλαδή το 604.

```
>>> 604//100
6
```

Τα σωστά βήματα είναι:

1. Διαιρείς τον αριθμό με το 1000 και κρατάς το ακέραιο μέρος
2. Αφαιρείς από τον αριθμό τις χιλιάδες που βρήκες
3. Διαιρείς τον αριθμό με το 100 και κρατάς το ακέραιο μέρος
4. Αφαιρείς από τον αριθμό τις εκατοντάδες που βρήκες
5. Διαιρείς τον αριθμό με το 10 και κρατάς το ακέραιο μέρος
6. Αφαιρείς από τον αριθμό τις δεκάδες που βρήκες
7. Σου μένουν οι μονάδες

Ας ονομάσουμε τον αριθμό 7604, n (n=7604), και το πρώτο ψηφίο, στην περίπτωσή μας τις χιλιάδες, prwto.

```
>>> n = 7604
>>> prwto = n//1000
>>> prwto
7
>>> n = n − prwto∗1000
>>> n
604
```

Ας δούμε λίγο αυτή την εντολή:

```
n = n − prwto∗1000
```

Θυμήσου ότι εκείνη τη στιγμή το n έχει την τιμή 7604 και το prwto την τιμή 7. Η παραπάνω εντολή σημαίνει:

1. Κάνε τις πράξεις που υπάρχουν δεξιά από το σύμβολο ίσον
2. Δώσε σαν τιμή στην μεταβλητή που υπάρχει αριστερά από το σύμβολο ίσον το αποτέλεσμα των πράξεων

Έτσι η Python κάνει πρώτο n−prwto∗1000 δηλαδή $7604 - 7 * 1000 = 604$ και αυτό το αποτέλεσμα το δίνει σαν τιμή στην μεταβλητή που υπάρχει αριστερά από το = δηλαδή στη μεταβλητή n. Έτσι το n τώρα είναι 604. *Προσοχή!* Η τιμή 7604 δεν υπάρχει στην μεταβλητή n. Με αυτόν τον τρόπο το n έχει πάντα τον αριθμό που χρειάζεσαι για να απομονώσεις το επόμενο ψηφίο του αριθμού.

Έτσι ένα συνολικό πρόγραμμα που μπορείς να γράψεις είναι:

```
n = 7604
prwto = n//1000
n = n − prwto∗1000
deutero = n//100
n = n − deutero∗100
trito = n //10
n = n − trito∗10
tetarto = n
print(prwto,end='')
print('∗10∗∗3+',end='')
print(deutero,end='')
print('∗10∗∗2+',end='')
print(trito,end='')
print('∗10+',end='')
print(tetarto)
```

Όταν το εκτελέσεις δίνει το σωστό αποτέλεσμα:

```
7∗10∗∗3+6∗10∗∗2+0∗10+4
```

Το παραπάνω πρόγραμμα δουλεύει με όλους τους τετραψήφιους αριθμούς, απλά άλλαξε το n σε όποιον αριθμό θέλεις στην αρχή του προγράμματος.

Όμως οι 7 εντολές print δεν είναι ο καλύτερος τρόπος να γράψεις το αποτέλεσμα. Θα ήταν καλύτερα να τυπώσουμε αυτά που πρέπει για κάθε ψηφίο χωριστά ώστε να έχουμε τέσσερις εντολές print ως εξής:

```
n = 7604
prwto = n//1000
n = n − prwto∗1000
deutero = n//100
n = n − deutero∗100
trito = n //10
n = n − trito∗10
tetarto = n
```

```
9 print(prwto + ’*10**3+’,end=’’)
10 print(deutero + ’*10**2+’,end=’’)
11 print(trito + ’*10+’,end=’’)
12 print(tetarto)
```

Εξάλλου όταν έχουμε πρόσθεση με λέξεις η Python τις βάζει δίπλα δίπλα οπότε για το print(prwto + ’*10**3+’,end=’’) θα περιμέναμε να εμφανιστεί το 7*10**3+. Άν όμως εκτελέσεις το παραπάνω πρόγραμμα θα προκύψει ένα μήνυμα λάθους.

```
Traceback (most recent call last):
  File ”a.py”, line 9, in <module>
    print(prwto + ’*10**3+’,end=’’)
TypeError: unsupported operand type(s) for +: ’int’ and ’str’
```

Αν προσέξουμε λίγο θα δούμε ότι το λάθος αφορά την ένατη γραμμή (line 9) και το λάθος είναι TypeError: unsupported operand type(s) for +: ’int’ and ’str’. Σε μετάφραση από τα αγγλικά το μήνυμα λάθους γράφει:

ΛάθοςΤύπων: μη υποστηριζόμενοι τύποι για το +: ’int’ και ’str’

Τι είναι οι τύποι και προκύπτουν λάθη από αυτούς;

Οτιδήποτε χρησιμοποιούμε στην Python έχει τύπο. Μάλιστα μπορούμε να δούμε τον τύπο αυτό με την εντολή type. Έτσι δοκίμασε:

```
>>> type(7)
<class ’int’>
>>> type(’a’)
<class ’str’>
>>> type(’7’)
<class ’str’>
```

Βλέπουμε ότι οι αριθμοί έχουν τύπο ‘int’, θα αγνοήσουμε τη λέξη class προς το παρόν. Ενώ οι λέξεις που έχουν τα εισαγωγικά έχουν τύπο ‘str’. Το ‘int’ προκύπτει από την αγγλική λέξη ‘integer’ που σημαίνει ακέραιος, και το ‘str’ προκύπτει από την αγγλική λέξη ‘string’ που σημαίνει μια σειρά από γράμματα και αριθμούς. Στα ελληνικά το ‘string’ το μεταφράζουμε ως αλφαριθμητικό.

Το πρόβλημα είναι ότι η Python δεν μπορεί να προσθέσει έναν ακέραιο με ένα αλφαριθμητικό. Γι’ αυτό δίνει και το μήνυμα λάθους. Ωστόσο, αυτό το πρόβλημα έχει λύση και είναι η μετατροπή του αριθμού σε αλφαριθμητικό με την εντολή str(). Δοκίμασε:

```
>>> type(7)
<class ’int’>
```

```
>>> str(7)
’7’
>>> type(’7’)
<class ’str’>
```

Το παραπάνω πρόγραμμα γίνεται λοιπόν:

```
n = 7604
prwto = n//1000
n = n − prwto*1000
deutero = n//100
n = n − deutero*100
trito = n //10
n = n − trito*10
tetarto = n
print(str(prwto) + ’*10**3+’,end=’’)
print(str(deutero) + ’*10**2+’,end=’’)
print(str(trito) + ’*10+’,end=’’)
print(str(tetarto))
```

Που και πάλι δίνει τη σωστή απάντηση.

Μπορούμε να βάλουμε τα στοιχεία prwto, deutero, trito και tetarto σε μια λίστα που θα την ονομάσουμε psifia και θα έχει τέσσερα στοιχεία. Στην αρχή θα αρχικοποιήσουμε τη λίστα με κάποια τιμή ειδικά στην περίπτωση που ξέρουμε πόσο μεγάλη θα είναι όπως τώρα.

```
n = 7604
psifia = [0,0,0,0]
psifia[0] = n//1000
n = n − psifia[0]*1000
psifia[1] = n//100
n = n − psifia[1]*100
psifia[2] = n //10
n = n − psifia[2]*10
psifia[3] = n
print(str(psifia[0]) + ’*10**3+’,end=’’)
print(str(psifia[1]) + ’*10**2+’,end=’’)
print(str(psifia[2]) + ’*10+’,end=’’)
print(str(psifia[3]))
```

Όπως βλέπεις το πρώτο ψηφίο είναι το psifia[0], το δεύτερο ψηφίο είναι το psifio[1] κ.λ.π., οι πίνακες στην Python ξεκινάν από το 0.

Φαίνεται ότι κάνουμε τα ίδια πράγματα τρεις φορές όπως βλέπεις εδώ:

```
psifia[0] = n//1000
n = n − psifia[0]∗1000
psifia[1] = n//100
n = n − psifia[1]∗100
psifia[2] = n //10
n = n − psifia[2]∗10
```

Θα προσπαθήσουμε να τα κάνουμε με for όπου το i θα μετράει 0,1,2 έτσι το psifia[0] θα γίνει psifia[i]. Όμως θα πρέπει να υπολογίσουμε το 1000 το 1000 είναι 10**3 και στην επόμενη επανάληψη είναι 10**2 κ.ο.κ. οπότε είναι 10**(3-i). Άρα οι τρεις παραπάνω εντολές μπορούν να αντικατασταθούν με μία for

```
for i in range(3):
    psifia[i] = n//10∗∗(3−i)
    n = n − psifia[i]∗ 10∗∗(3−i)
```

Το ίδιο πρέπει να γίνει και με τις τρεις εντολές print:

```
print(str(psifia[0]) + ’∗10∗∗3+’,end=’’)
print(str(psifia[1]) + ’∗10∗∗2+’,end=’’)
print(str(psifia[2]) + ’∗10+’,end=’’)
```

Θα πρέπει να κάνουμε έναν ιδιαίτερο χειρισμό για τις δυνάμεις όπου θα πρέπει να μειώνονται καθώς συνέχίζουν οι επαναλήψεις οπότε αντί για ‘*10**3’ χρειαζόμαστε ‘*10**(3-i)+’ όμως το 3-i είναι αριθμός οπότε για να το γράψουμε στην Python θα χρειαστεί να βάλουμε το str() και να γίνει ’∗10∗∗’+str(3−i)+’+’. Οι παραπάνω τρεις εντολές μπορούν τώρα να γραφούν με μία for ως εξής:

```
for i in range(3):
    print(str(psifia[i]) + ’∗10∗∗’ + str(3−i) + ’+’,end=’’)
```

και όλο το πρόγραμμα να γίνει:

```
n = 7604
psifia = [0,0,0,0]
for i in range(3):
    psifia[i] = n//10∗∗(3−i)
    n = n − psifia[i]∗ 10∗∗(3−i)
```

```
psifia[3] = n
for i in range(3):
    print(str(psifia[i]) + '*10**' + str(3-i) + '+',end='')
print(str(psifia[3]))
```

Πώς μπορεί το πρόγραμμα αυτό να δουλεύει για όλους τους ακέραιους ανεξάρτητα από το μέγεθός τους; Αν μετατρέψουμε τον ακέραιο σε αλφαριθμητικό η Python μπορεί να μας πει πόσο μεγάλος είναι με την εντολή len, που είναι το μήκος του αλφαριθμητικού.

```
>>> n = 7604
>>> len(str(n))
4
>>> n = 7102234
>>> len(str(n))
7
```

Και η αρχικοποίηση του πίνακα μπορεί να γίνει για όσα ψηφία θέλουμε (plithos) με την εντολή [0]*plithos:

```
>>> plithos = 7
psifia = [0] * plithos
>>> psifia
[0, 0, 0, 0, 0, 0, 0]
```

Αν υπολογίσουμε το plithos των ψηφίων με το len(str(n)) θα προκύψει 4, ωστόσο εμείς στο πρόγραμμά μας κάνουμε επαναλήψεις τρεις φορές γιατί χρειαζόμαστε ειδικό χειρισμό στο τελευταίο ψηφίο. Έτσι αντικαθιστούμε το 3 με plithos-1 παντού στο πρόγραμμα.

```
n = 7604
plithos = len(str(n))
psifia = [0]*plithos
for i in range(plithos-1):
    psifia[i] = n//10**(plithos-1-i)
    n = n - psifia[i]* 10**(plithos-1-i)
psifia[plithos-1] = n
for i in range(plithos-1):
    print(str(psifia[i]) + '*10**' + str(plithos-1-i) + '+',end='')
print(str(psifia[plithos-1]))
```

Μπορούμε να το γράψουμε με τη μορφή συνάρτησης ως εξής:

```
def anaptygma(n):
  plithos = len(str(n))
  psifia = [0]*plithos
  for i in range(plithos−1):
        psifia[i] = n//10**(plithos−1−i)
        n = n − psifia[i]* 10**(plithos−1−i)
  psifia[plithos−1] = n
  for i in range(plithos−1):
        print(str(psifia[i]) + '*10**' + str(plithos−1−i) + '+',end='')
  print(str(psifia[plithos−1]))

>>> anaptygma(7604)
7*10**3+6*10**2+0*10**1+4*10**0
>>> anaptygma(7102234)
7*10**6+1*10**5+0*10**4+2*10**3+2*10**2+3*10**1+4
```

Η Python έχει διάφορους τρόπους να συμπυκνώνει μεγάλα προγράμματα ακόμη και σε μία γραμμή. Ένας τέτοιος τρόπος να γραφτεί το ανάπτυγμα είναι ο εξής:

```
>>> n = 7604
>>> '+'.join([x+'*10**'+str(len(str(n))−1−i)
for (i,x) in enumerate(list(str(n)))])

'7*10**3+6*10**2+0*10**1+4*10**0'
```

1.10 Ιστορικό σημείωμα

(Στο βιβλίο βρίσκεται στη Σελ. 17)

Όταν ο δάσκαλος ζήτησε από τους υπόλοιπους μαθητές να υπολογίσουν το άθροισμα $1 + 2 + 3 + \ldots + 98 + 99 + 100$, πριν οι υπόλοιποι αρχίσουν τις πράξεις ο μικρός Γκάους το είχε ήδη υπολογίσει. Ο δάσκαλος ρώτησε έκπληκτος πώς το βρήκε. Τότε εκείνος έγραψε στον πίνακα: Ο Γκάους σκέφτηκε πως το άθροισμα

$$1 + 2 + 3 + \ldots + 99 + 100$$

είναι ίδιο με το

$$100 + 99 + 98 + \ldots + 2 + 1$$

και αν αθροίσουμε τον πρώτο όρο με τον πρώτο όρο, τον δεύτερο με τον δεύτερο κ.ο.κ. Συνολικά αυτό γίνεται:

$$\underbrace{(1+100)+(2+99)+\ldots(99+2)+(100+1)}_{50\text{φορές}} = \tag{1.1}$$

$$101 \cdot 100 \tag{1.2}$$

$$\tag{1.3}$$

Άρα το άθροισμα $1+2+3+\ldots+99+100$ είναι $\frac{1}{2}101 \times 100 = 5050$.

Μπορείς να υπολογίσεις το άθροισμα $1+2+3+\ldots+999+1000$ με τον τρόπο του Γκάους;

Ας δούμε το πρόβλημα από την πλευρά της Python. Μπορούμε να εμφανίζουμε τους αριθμούς από το 1 έως το 100 με το παρακάτω πρόγραμμα:

```
for i in range(101):
    print(i)
```

Πώς όμως θα αθροίσουμε τους αριθμούς αυτούς. Θα φτάξουμε μιά νέα μεταβλητή athroisma και σε αυτή θα προσθέτουμε το i κάθε φορά. Το πρόγραμμα γίνεται:

```
athroisma = 0
for i in range(101):
    athroisma = athroisma + i
print(athroisma)
```

Όμως επειδή το άθροισμα είναι χρήσιμο σε πολλές περιπτώσεις η Python έχει έτοιμο το άθροισμα με την εντολή sum. Δοκίμασε:

```
>>> sum([1,2,3])
6
```

Με τον ίδιο τρόπο μπορείς να βρεις το άθροισμα $1+2+3+\ldots+99+100$:

```
>>> sum(range(101))
5050
```

Τέλος ο τρόπος του μικρού Γκάους είναι:

```
>>> 101*100/2
5050
```

Με την Python μπορούμε να υπολογίσουμε το άθροισμα από το 1 έως το 1000 και με τους δύο τρόπους:

```
>>> sum(range(1001))
500500
>>> 1000*1001/2
500500.0
```

Στην Python to 500500.0 σημαίνει πως το δεκαδικό μέρος είναι 0 οπότε το αποτέλεσμα είναι και πάλι 500500.