ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

Εισαγωγή στους Υπολογιστές

Εργαστήριο 5

Καθηγητές: Αβούρης Νικόλαος, Παλιουράς Βασίλης, Κουκιάς Μιχαήλ, Σγάρμπας Κυριάκος

Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

ΑΝΟΙΚΤΑ ακαδημαϊκά μαθήματα ΠΠ

Εργαστήριο 5: Δεύτερη Άσκηση Προγραμματισμού Python

5.1 Γενικά

Στην άσκηση αυτή θα εξεταστούν μερικά πιο εξελιγμένα χαρακτηριστικά της γλώσσας Python, όπως είναι οι επαναλήψεις, ο ορισμός συναρτήσεων από τον προγραμματιστή, καθώς και μερικές συναρτήσεις γραφικών.

5.2 Ορισμός Συναρτήσεων

Στην προηγούμενη άσκηση είδαμε συνοπτικά τον τρόπο χρήσης συναρτήσεων από τις βιβλιοθήκες της Python με την εντολή import. Τώρα θα δούμε πως μπορούμε να ορίσουμε δικές μας συναρτήσεις. Έστω η συνάρτηση $f(x)= 0.0001*x^3 + 0.015*x^2 - 1.14*x - 2.8$. Στην Python την ορίζουμε με την εντολή def ως εξής:

```
>>> def f(x):

return 0.0001*x**3+0.015*x**2-1.14*x-2.8
```

Δηλαδή, μετά την def βάζουμε το όνομα της συνάρτησης και σε παρένθεση την ανεξάρτητη μεταβλητή. Η εντολή return στην επόμενη καθορίζει την τιμή που θα 'επιστρέψει' η συνάρτηση μόλις την καλέσουμε (για παράδειγμα, η τιμή που θα τυπώνει αν την εκτελέσουμε με την print). Τώρα μπορούμε να την καλέσουμε ακριβώς όπως τις ενσωματωμένες συναρτήσεις της γλώσσας:

```
>>> print f(3.5)
-6.6019625
>>> print f(-12), f(111.11)
12.8672 192.886177563
```

Αν στη συνάρτηση υπάρχουν περισσότερες από μια ανεξάρτητες μεταβλητές (λέγονται και **ορίσματα**), τις χωρίζουμε με κόμματα. Για παράδειγμα, να μια συνάρτηση που υπολογίζει τη διακρίνουσα ενός τριωνύμου από τους συντελεστές a, b και c:

```
>>> def diakrinousa(a,b,c):
        return b*b-4*a*c
>>> print diakrinousa(4,2,-1)
20
```

Επιπλέον, οι συναρτήσεις μπορούν να αποτελούνται από πολλές γραμμές κώδικα, να καλούν άλλες συναρτήσεις και δεν είναι απαραίτητο να επιστρέφουν μια αριθμητική τιμή αλλά οτιδήποτε. Δείτε το παρακάτω παράδειγμα:

```
>>> def remainder(x,y):
        z=x/y
```

```
        r=x-z*y
        return r

>>> def check_odd_or_even(x):
        y=remainder(x,2)
        if y==0:
                return 'even'
        else:
                return 'odd'
```

Η συνάρτηση remainder: υπολογίζει και επιστρέφει το υπόλοιπο της διαίρεσης δύο ακέραιων αριθμών. Στη συνέχεια η check_odd_or_even καλεί την remainder για να διαπιστώσει αν κάποιος ακέραιος είναι περιττός (odd) ή άρτιος (even).

```
>>> print remainder(8,3)
2
>>> print check_odd_or_even(13)
odd
>>> print check_odd_or_even(14)
even
```

Προσέξτε ότι όπως γράψαμε την check_odd_or_even επιστρέφει αλφαριθμητικά (λέξεις).

5.3 Επαναλήψεις με for και Λίστες τιμών

Έστω ότι θέλουμε να τυπώσουμε τις τιμές της συνάρτησης f(x) που ορίσαμε προηγουμένως για πολλές τιμές του x. Για να αποφύγουμε να δώσουμε πολλές εντολές print, η Python επιτρέπει να γράψουμε κάτι τέτοιο:

```
>>> for x in [0, 1, 2, 3, 4]:
        print 'Για x=', x, '    f(x)=',f(x)

Για x= 0    f(x)= -2.8
Για x= 1    f(x)= -3.9249
Για x= 2    f(x)= -5.0192
Για x= 3    f(x)= -6.0823
Για x= 4    f(x)= -7.1136
```

Το [0, 1, 2, 3, 4] λέγεται λίστα. Περιέχει πέντε αριθμούς και η εντολή for θα δώσει διαδοχικά στη μεταβλητή x κάθε τιμή που βρίσκεται μέσα στη λίστα και θα επαναλάβει τις εσωτερικές της εντολές (εδώ έχει μόνο την print) για κάθε τιμή του x. Το χρήσιμο με αυτή τη σύνταξη είναι ότι η Python διαθέτει την ενσωματωμένη συνάρτηση range που δημιουργεί λίστες αριθμών και δεν χρειάζεται να τους πληκτρολογούμε εκ των προτέρων:

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

Η range όταν καλείται με ένα όρισμα επιστρέφει μια λίστα με όλους τους ακεραίους που δεν ξεπερνούν το όρισμα, ξεκινώντας από το 0.

```
>>> print range(-3,4)
[-3, -2, -1, 0, 1, 2, 3]
```

Αν την καλέσουμε με δυο ορίσματα καθορίζουμε εμείς από που θα ξεκινήσει.

```
>>> print range(1,15,2)
[1, 3, 5, 7, 9, 11, 13]
```

Με τρία ορίσματα καθορίζουμε και το βήμα με το οποίο θα μετράει. Εδώ επέστρεψε όλους τους ακεραίους που είναι μικρότεροι από 15, ξεκινώντας από το 1 και μετρώντας ανά 2.

Έτσι, αν θέλουμε να γίνουν πολλές επαναλήψεις, μπορούμε να συνδυάσουμε την συνάρτηση range() με την εντολή for, γράφοντας κάτι τέτοιο:

```
>>> for x in range(100):
        print 'Για x=', x, '   f(x)=',f(x)

Για x= 0    f(x)= -2.8
Για x= 1    f(x)= -3.9249
Για x= 2    f(x)= -5.0192
Για x= 3    f(x)= -6.0823
...
Για x= 97    f(x)= 119.0223
Για x= 98    f(x)= 123.6592
Για x= 99    f(x)= 128.3849
```

Μπορούμε να έχουμε επαναλήψεις ως εσωτερικές εντολές σε άλλες επαναλήψεις. Σε αυτή την περίπτωση οι επαναλήψεις λέγονται εμφωλευμένες (nested). Εμφωλευμένες επαναλήψεις χρησιμοποιούνται συχνά για να αναζητήσουμε λύσεις σε κάποιο πρόβλημα με εξαντλητικό τρόπο, αρκεί να γνωρίζουμε τη συνθήκη που πρέπει να ικανοποιηθεί και το πεδίο ορισμού του είναι αρκετά περιορισμένο. Για παράδειγμα:

Πρόβλημα 1: Να βρεθούν όλες οι πυθαγόρειες τριάδες ακεραίων (a, b, c) με 0<a<b<c<50.

Πυθαγόρειες λέγονται οι τριάδες ακεραίων που ικανοποιούν την συνθήκη του πυθαγόρειου θεωρήματος, δηλαδή $a^2+b^2=c^2$.

Αφού οι αριθμοί πρέπει να είναι ακέραιοι στο διάστημα (0,50) το πεδίο ορισμού είναι αρκετά περιορισμένο και μπορούμε να κάνουμε εξαντλητική αναζήτηση μέσα σε αυτό (δηλαδή να εξετάσουμε όλους τους δυνατούς συνδυασμούς) γράφοντας κάτι τέτοιο:

```
>>> for a in range(1,50):
        for b in range(a+1,50):
            for c in range(b+1,50):
                if a*a+b*b==c*c:
                    print a, b, c

345
5 12 13
6 8 10
7 24 25
8 15 17
9 12 15
9 40 41
10 24 26
12 16 20
12 35 37
15 20 25
```

```
15 36 39
16 30 34
18 24 30
20 21 29
21 28 35
24 32 40
27 36 45
```

Οι Εμφωλευμένες εντολές for εξασφαλίζουν ότι συνδυάζουμε κάθε τιμή του a με όλες τις τιμές του b και κάθε τιμή του b με όλες τις τιμές του c. Προσέξτε επίσης ότι ξεκινάμε το range της κάθε μιας έναν ακέραιο μετά την τιμή της προηγούμενης μεταβλητής ώστε να εξασφαλίσουμε ότι a<b<c.

Επαναλήψεις πολύ συχνά χρησιμοποιούνται στο εσωτερικό συναρτήσεων, όπως στο επόμενο παράδειγμα:

Πρόβλημα 2: Να γραφτεί συνάρτηση που θα δέχεται έναν ακέραιο αριθμό και θα εξετάζει αν είναι πρώτος ή όχι.

Πρώτος (prime) λέγεται ένας θετικός ακέραιος αν δεν διαιρείται ακριβώς με κανέναν άλλον (θετικό) ακέραιο (εκτός από τον εαυτό του και τη μονάδα). Ένας ισοδύναμος ορισμός λέει ότι πρώτος είναι ένας ακέραιος όταν έχει ακριβώς δυο διαιρέτες (για αυτό το λόγο ο 1 δεν θεωρείται πρώτος).

Απευθείας από τον πρώτο ορισμό μπορούμε να γράψουμε:

```
>>> def is_prime(n):
        if n<2:
            return('no')
        for x in range(2,n-1):
            if remainder(n,x)==0:
                return('no')
        return('yes')
```

Εδώ χρησιμοποιήθηκε επαναληπτικά η συνάρτηση remainder() για να υπολογίσει τα υπόλοιπα όλων των διαιρέσεων του δοσμένου αριθμού n με κάθε αριθμό από 2 μέχρι n-1. Αν κάποιο από αυτά είναι μηδέν τότε ο αριθμός δεν είναι πρώτος και η συνάρτηση επιστρέφει 'no'. Αν το for τελειώσει χωρίς να μηδενιστεί κανένα υπόλοιπο τότε επιστρέφει 'yes'. Τώρα μπορούμε να τη χρησιμοποιήσουμε:

```
>>> for t in range(20):
        print t, is_prime(t)
```

```
0 no
1 no
2 yes
3 yes
4 no
5 yes
6 no
7 yes
8 no
9 no
10 no
11 yes
12 no
13 yes
14 no
15 no
16 no
17 yes
18 no
19 yes
```

Σημειωτέον ότι ο παραπάνω αλγόριθμος υπολογισμού δεν είναι ιδιαίτερα γρήγορος. Η καθυστέρησή του είναι εμφανής όσο μεγαλώνουν οι αριθμοί που ελέγχει. Πιο γρήγοροι αλγόριθμοι μπορούν να γραφτούν αν λάβουμε υπόψη ότι οι διαιρέτες των αριθμών εμφανίζονται κατά ζεύγη (δηλαδή αν ο a είναι διαιρέτης του n, τότε και ο n/a θα είναι διαιρέτης του n) και ότι αν ένας αριθμός δεν είναι διαιρέτης τότε κανένα πολλαπλάσιό του δεν θα είναι διαιρέτης.

5.4 Συναρτήσεις Γραφικών

Η Python διαθέτει πολλές βιβλιοθήκες γραφικών. Ίσως η πιο απλή στη χρήση είναι η βιβλιοθήκη γραφικών χελώνας (turtle graphics). Πήρε το όνομά της από τη γλώσσα προγραμματισμο Logo, που στις αρχές της δεκαετίας του 1980 εισήγαγε ένα παρόμοιο σύστημα γραφικών που στη θέση του κέρσορα είχε σχεδιασμένη μια χελώνα στην οποία ο προγραμματιστής έδινε εντολές να μετακινηθεί και να σχεδιάσει γραμμές πάνω στην οθόνη.

Για να φορτώσουμε τη βιβλιοθήκη δίνουμε την εντολή:

```
>>> from turtle import *
```

Και αμέσως μετά, αν δώσουμε κάποια εντολή[1] γραφικών όπως:

```
>>> reset()
```

Εμφανίζεται ένα νέο παράθυρο με όνομα «Turtle Graphics»:

Το παράθυρο είναι άδειο αλλά ακριβώς στο κέντρο του έχει σχεδιασμένη μια γραφίδα (ή cursor, ή «χελώνα») η οποία περιμένει τις εντολές μας για να σχεδιάσει γραφικά πάνω στο νέο παράθυρο. Η εντολή

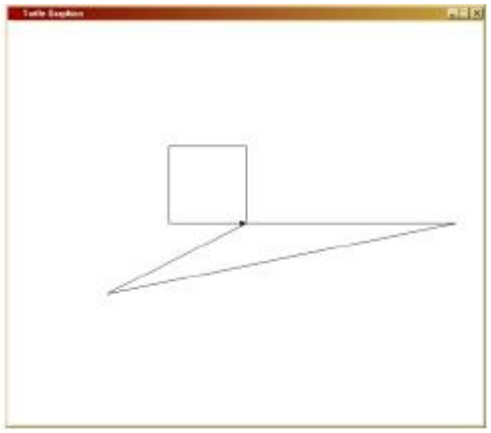
[1] Αν και αναφερόμαστε σε "εντολές γραφικών" καλό είναι να θυμόμαστε ότι στην πραγματικότητα πρόκειται για συναρτήσεις. Για αυτό και ακόμα κι όσες δεν δέχονται κανένα όρισμα τις γράφουμε με παρενθέσεις στο τέλος του ονόματός τους.

reset σβήνει ότι τυχόν έχουμε σχεδιάσει προηγουμένως στο παράθυρο και τοποθετεί τη γραφίδα στο κέντρο με προσανατολισμό προς τα δεξιά. Το κέντρο έχει συντεταγμένες (0,0). Μπορούμε να μετακινήσουμε τη γραφίδα σε κάποιες άλλες συντεταγμένες με την εντολή goto. Πχ.:

```
>>> goto(300,0)
```

Η γραφίδα μετακινήθηκε κατά 300 pixels (εικονοστοιχεία ή ψηφίδες) προς τα δεξιά αφήνοντας ως ίχνος μια ευθεία γραμμή. Μπορούμε να δώσουμε οποιοδήποτε σημείο ως παράμετρο στην εντολή goto (ακόμα κι αν αυτό βρίσκεται εκτός του παραθύρου). Με διαδοχικές εντολές goto μπορούμε να σχεδιάσουμε απλά σχήματα:

```
>>> goto(-200,-100)
>>> goto(0,0)
>>> goto(0,110)
>>> goto(-110,110)
>>> goto(-110,0)
>>> goto(0,0)
```

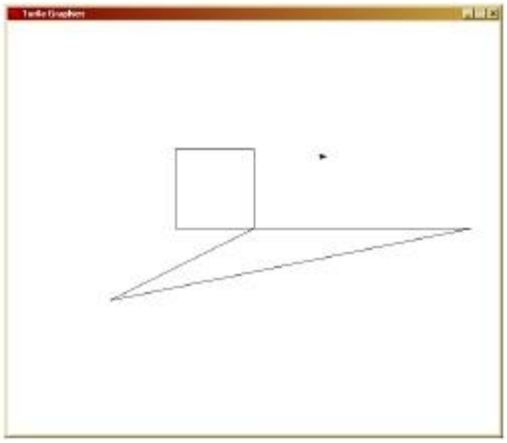


Αν η ταχύτητα με την οποία σχεδιάζονται οι γραμμές είναι αργή μπορείτε να δώσετε την εντολή:

```
>>> speed('fastest')
```

... και στο εξής όλες οι γραμμές θα σχεδιάζονται στη μέγιστη ταχύτητα. Αν θέλουμε να μεταβούμε σε κάποιο σημείο χωρίς να αφήσουμε ίχνος, δίνουμε πρώτα την εντολή up, πχ:

```
>>> up()
>>> goto(100,100)
```



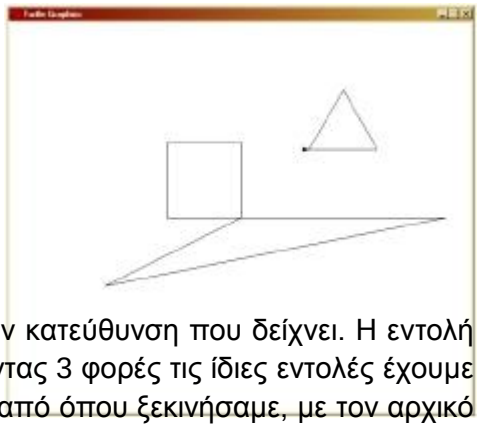
Και μόλις πάμε στο σημείο που θέλουμε δίνουμε:

```
>>> down()
```

... για να αφήνουμε στο εξής ίχνος. Φανταστείτε ότι σηκώνουμε (up) το στυλό από το χαρτί μας, τον μετακινούμε (goto) σε ένα άλλο σημείο και μετά τον κατεβάζουμε (down) στο χαρτί για να συνεχίσουμε να σχεδιάζουμε.

Στον μέχρι τώρα σχεδιασμό χρησιμοποιήσαμε καρτεσιανές συντεταγμένες. Όμως η βιβλιοθήκη διαθέτει και συναρτήσεις για σχεδιασμό με πολικές συντεταγμένες (στις οποίες δίνουμε γωνία και απόσταση. Για παράδειγμα, να πώς σχεδιάζουμε ένα ισόπλευρο τρίγωνο πλευράς 100 pixels με πολικές συντεταγμένες:

```
>>> forward(100)
>>> left(120)
>>> forward(100)
>>> left(120)
>>> forward(100)
>>> left(120)
```

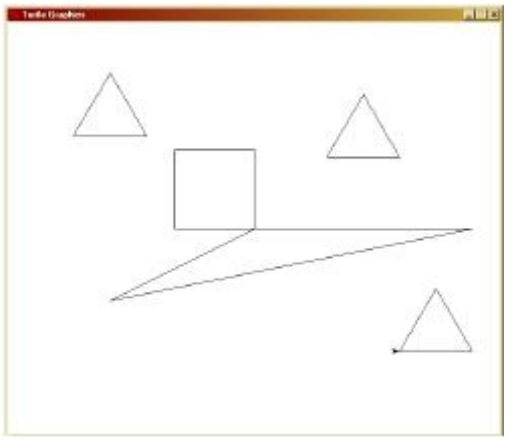


Η εντολή forward (100) μετακινεί τη γραφίδα 100 pixels προς την κατεύθυνση που δείχνει. Η εντολή left (120) στρίβει τη γραφίδα 120 μοίρες προς τα αριστερά. Δίνοντας 3 φορές τις ίδιες εντολές έχουμε φτιάξει ένα ισόπλευρο τρίγωνο και έχουμε καταλήξει στο σημείο από όπου ξεκινήσαμε, με τον αρχικό προσανατολισμό:

Το καλό όταν σχεδιάζουμε σε πολικές συντεταγμένες είναι ότι με τις ίδιες εντολές φτιάχνουμε ακριβώς τα ίδια σχήματα σε οποιοδήποτε σημείο της οθόνης κι αν βρισκόμαστε:

```
>>> up()
>>> goto(-250,130)
>>> down()
>>> forward(100)
>>> left(120)
>>> forward(100)
>>> left(120)
```

```
>>> forward(100)
>>> left(120)
>>> up()
>>> goto(200,-170)
>>> down()
>>> forward(100)
>>> left(120)
>>> forward(100)
>>> left(120)
>>> forward(100)
>>> left(120)
```



Για να κάναμε το ίδιο με καρτεσιανές συντεταγμένες (δηλαδή με εντολές goto) θα ήταν αρκετά πιο δύσκολο αφού θα έπρεπε να υπολογίσουμε τις ακριβείς συντεταγμένες των κορυφών των τριγώνων.

Η βιβλιοθήκη turtle περιέχει και άλλες χρήσιμες εντολές που μπορούμε να δούμε στο on-line help[2] της Python:

Κάποιες από αυτές θα χρησιμοποιήσουμε στη συνέχεια. Προς το παρόν ξέρουμε αρκετές εντολές για να κάνουμε κάτι χρήσιμο όπως τον σχεδιασμό γραφικών παραστάσεων μαθηματικών συναρτήσεων.

5.5 Γραφικές Παραστάσεις

Στην ενότητα 5.2 είδαμε πώς ορίζουμε μαθηματικές συναρτήσεις με το παράδειγμα της f(x)=0.0001*x3+0.015*x2-1.14*x-2.8 :

```
>>> def f(x):
        return 0.0001*x**3+0.015*x**2-1.14*x-2.8
```

[2] Όπως μπορείτε να δείτε στο on-line-help, κάποιες συναρτήσεις έχουν διπλά ονόματα, πχ. αντί για forward(100) μπορούμε να γράψουμε fd(100). Προς το παρόν αγνοήστε το πρόθεμα "turtle", που εμφανίζεται στο help μπροστά από τα ονόματα των συναρτήσεων. Αυτό χρειάζεται αν καλέσουμε με διαφορετικό τρόπο την import και θα το εξηγήσουμε σε επόμενο εργαστήριο.

Τώρα θα δούμε πώς να κάνουμε γραφικές παραστάσεις συναρτήσεων με τις εντολές που μάθαμε. Δίνουμε τις εντολές:

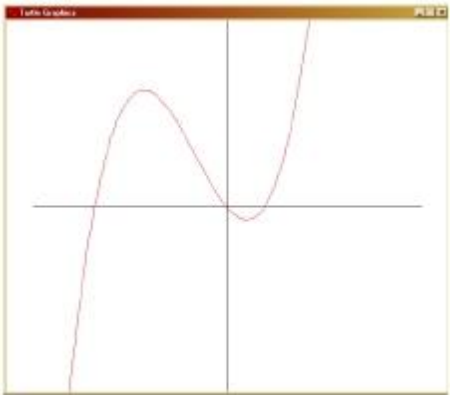
```
>>> reset()
>>> speed('fastest')
>>> forward(300)
>>> goto(0,0)
>>> left(90)
>>> forward(300)
>>> goto(0,0)
>>> left(90)
>>> forward(300)
>>> goto(0,0)
>>> left(90)
>>> forward(300)
>>> up()
>>> goto(-300,f(-300))
>>> down()
```

Οι παραπάνω εντολές σχεδιάζουν τους δυο άξονες και μετακινούν τη γραφίδα στο σημείο (-300,f(-300)) από όπου θα ξεκινήσουμε τη σχεδίαση. Ας χρησιμοποιήσουμε κόκκινο χρώμα για τη σχεδίαση. Γράφουμε:

```
>>> color('red')
```

... και στη συνέχεια:

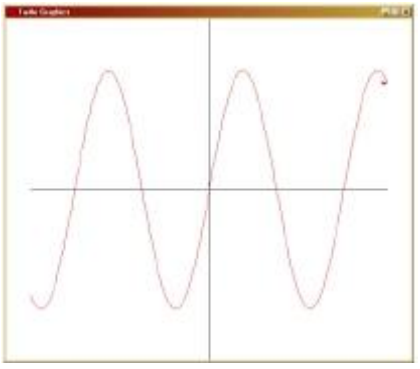
```
>>> for x in range(-300,300):
        goto(x,f(x))
```



Και αυτή είναι η γραφική παράσταση της f(x) στο διάστημα [-300,300].

Όμως η συγκεκριμένη συνάρτηση «έτυχε» να φαίνεται καλά στο παράθυρο. Πολύ συχνά η συνάρτηση χρειάζεται αλλαγή κλίμακας είτε στον άξονα x είτε στον άξονα y για να εμφανιστεί σωστά. Φανταστείτε για παράδειγμα να προσπαθούσαμε να απεικονίσουμε τη συνάρτηση του ημιτόνου, sin(x). Μια που το πλάτος της θα ήταν μόλις ένα pixel και η περίοδος 6 pixels, δεν θα βλέπαμε τίποτε. Χρησιμοποιώντας αλλαγή κλίμακας όμως, μπορούμε να γράψουμε κάτι τέτοιο:

```
>>> from math import *
>>> cy=200
>>> cx=36.
>>> x=-300
>>> up()
>>> goto(x,cy*sin(x/cx))
>>> down()
>>> for x in range(-300,300):
        goto(x,cy*sin(x/cx))
```



Τα cx, cy είναι οι μεγεθύνσεις κατά τους άξονες των x και y αντίστοιχα. Πλέον το πλάτος είναι 200 pixels και η περίοδος 2π*36=226 pixels και η γραφική παράσταση της συνάρτησης είναι ευδιάκριτη στο παράθυρο.

5.6 Αλγοριθμική Ζωγραφική

Η βιβλιοθήκη turtle διαθέτει τη συνάρτηση circle(r) η οποία σχεδιάζει έναν κύκλο ακτίνας r. Ας φτιάξουμε ένα πρόγραμμα που σχεδιάζει 100 κύκλους τυχαίων ακτινών σε τυχαία σημεία με τυχαία χρώματα κι ας δούμε πόσο ωραίο θα φανεί.

Θα χρειαστούμε τη βιβλιοθήκη random που περιέχει διάφορες συναρτήσεις για την παραγωγή (ψευδο)τυχαίων αριθμών:

```
>>> from random import *
```

Η βιβλιοθήκη random περιέχει μια συνάρτηση random() η οποία κάθε φορά που καλείται επιστρέφει έναν τυχαίο πραγματικό αριθμό μεταξύ 0 και 1:

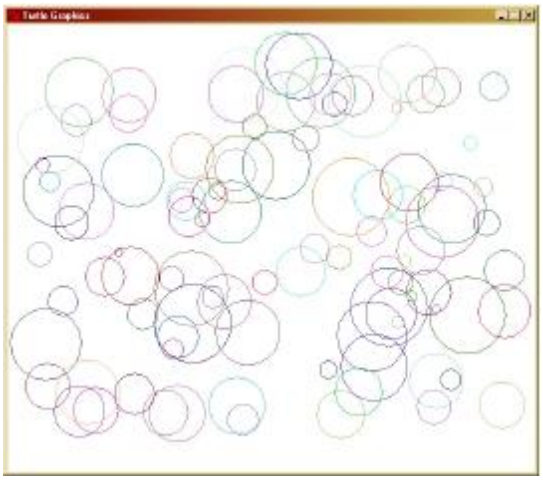
```
>>> print random()
0.178361201055
>>> print random()
0.671443452792
```

Περιέχει επίσης και τη συνάρτηση randint(low,high) η οποία κάθε φορά που καλείται επιστρέφει έναν ακέραιο αριθμό στο διάστημα [low,high]:

```
>>> print randint(10,100)
47
```

Να πώς τις χρησιμοποιούμε:

```
>>> for i in range(100):
        up()
        x=randint(-300,300)
        y=randint(-250,200)
        goto(x,y)
        down()
        color(random(),random(),random())
        r=randint(5,50)
        circle(r)
```



Στο παραπάνω τμήμα κώδικα μετακινούμε τη γραφίδα σε ένα τυχαίο σημείο (x,y) όπου το x παίρνει τιμές στο διάστημα [-300,300] και το y παίρνει τιμές στο διάστημα [-250,200]. Στη συνέχεια επιλέγουμε ένα τυχαίο χρώμα μέσω της συνάρτησης color(R,G,B). Η συνάρτηση αυτή είναι διαφορετική από την color που χρησιμοποιήσαμε στις γραφικές παραστάσεις. Αντί να καθορίσουμε το χρώμα με το όνομά του, δίνουμε τρία ορίσματα που αντιστοιχούν στις ποσότητες των τριών βασικών χρωμάτων (κόκκινο, πράσινο, μπλε) που θα αναμιχθούν για να προκύψει το χρώμα του κύκλου. Οι αριθμοί είναι πραγματικοί στο διάστημα [0,1] και οι συνδυασμοί τους παράγουν όλα τα δυνατά χρώματα. Τέλος καθορίζεται η ακτίνα του κύκλου r ως τυχαίος ακέραιος στο διάστημα [5,50] και δίνεται εντολή να σχεδιαστεί ο κύκλος. Η διαδικασία επαναλαμβάνεται 100 φορές αφού βρίσκεται στο εσωτερικό της for και το αποτέλεσμα φαίνεται στην διπλανή εικόνα

Τι θα αλλάζαμε στον κώδικα αν θέλαμε αντί για κύκλους να σχεδιάσουμε τρίγωνα; Προφανώς θα έπρεπε να αντικαταστήσουμε την συνάρτηση circle(r) με κάποια άλλη που σχεδιάζει τρίγωνα. Αλλά

αν ψάξουμε στο help της βιβλιοθήκης turtle, δεν θα βρούμε τέτοια συνάρτηση. Μπορούμε όμως να ορίσουμε εμείς μια. Ήδη στην ενότητα 5.4 είδαμε πώς να φτιάχνουμε ισόπλευρα τρίγωνα.

Χρησιμοποιώντας τις ίδιες εντολές να μια συνάρτηση που σχεδιάζει ένα ισόπλευρο τρίγωνο πλευράς r:

```
>>> def triangle(r):
        forward(r)
        left(120)
        forward(r)
        left(120)
        forward(r)
        left(120)
```

Και τώρα:

```
>>> clear()
```

Καθαρίζουμε πρώτα την οθόνη για να μη ζωγραφίσουμε πάνω στους κύκλους. Η συνάρτηση clear() αντίθετα με τη reset() καθαρίζει την οθόνη χωρίς να αλλάξει τη θέση της γραφίδας, την ταχύτητά της ή το χρώμα.

```
>>> for i in range(100):
        up()
        x=randint(-300,300)
        y=randint(-250,200)
        goto(x,y)
        down()
        color(random(),random(),random())
        r=randint(5,50)
        triangle(r)
```

Και τώρα ας τα συνδυάσουμε. Έστω ότι θέλουμε να ζωγραφίσουμε 100 σχήματα που θα αριθμούνται από 0 έως 99. Αν ο αύξων αριθμός είναι πρώτος (θα ελέγχεται με τη συνάρτηση is_prime που ορίσαμε νωρίτερα) το σχήμα που θα σχεδιάζεται θα είναι κύκλος. Διαφορετικά θα σχεδιάζεται τρίγωνο. Για να ελέγξουμε την ορθότητα του αλγορίθμου θα γράφουμε δίπλα σε κάθε σχήμα τον αύξοντα αριθμό του. Αυτό το κάνει η συνάρτηση write(). Ακολουθεί ο κώδικας:

```
>>> clear()
>>> for i in range(100):
        up()
        x=randint(-300,300)
        y=randint(-250,200)
        goto(x,y)
        down()
        color(random(),random(),random())
        r=randint(5,50)
        if is_prime(i)=='yes':
                circle(r)
        else:
                triangle(r)
        write(i)
```

Αν παρακολουθήσατε τον τρόπο με τον οποίο σχεδιάζονται οι κύκλοι, ίσως παρατηρήσατε ότι σχεδιάζονται πάντα με ανθωρολογιακή φορά, εφαπτόμενοι στο διάνυσμα που ορίζει η γραφίδα στο σημείο που βρίσκεται. Η συνάρτηση circle μπορεί να δεχτεί και δεύτερο όρισμα που καθορίζει το τόξο σε μοίρες που θα σχεδιαστεί. Για παράδειγμα, η συνάρτηση:

```
>>> circle(50,360)
```

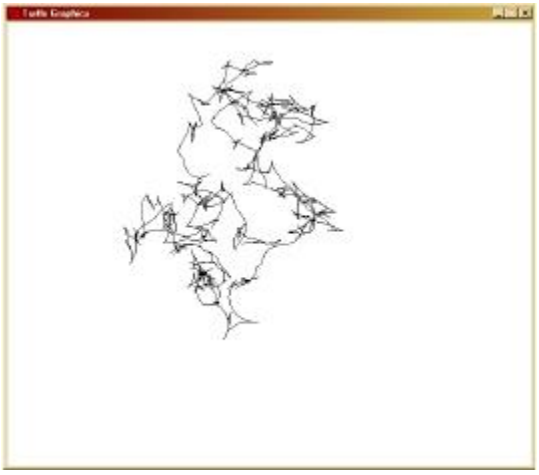
… θα σχεδιάσει έναν πλήρη κύκλο (τόξο 360 μοιρών) με ακτίνα 50, όμως η:

```
>>> circle(50,180)
```

... θα σχεδιάσει ένα ημικύκλιο (τόξο 180 μοιρών). Επιπλέον, η circle μπορεί να δεχτεί και αρνητικούς αριθμούς. Αρνητική γωνία σημαίνει ότι θα σχεδιάσει το τόξο με ωρολογιακή φορά.

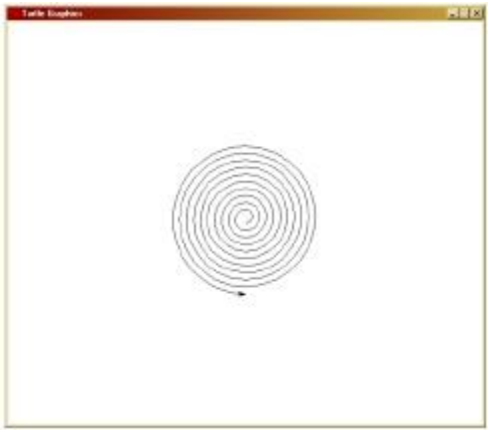
Αρνητική ακτίνα σημαίνει ότι θα φτιάξει το κατοπτρικό τόξο (υπάρχουν πάντα δυο εφαπτόμενα τόξα στο δεδομένο σημείο της γραφίδας). Δείτε για παράδειγμα αυτό το τμήμα κώδικα:

```
>>> reset()
>>> speed('fastest')
>>> for i in range(1000):
        circle(randint(-40,40),randint(-45,45))
```



Σχεδιάζει 1000 διαδοχικά τυχαία τόξα, με ακτίνες που δεν ξεπερνούν τα 40 pixels και γωνίες που δεν ξεπερνούν τις 45 μοίρες. Μπορούμε να χρησιμοποιήσουμε διαδοχικά τόξα για να φτιάξουμε σπείρες. Για παράδειγμα:

```
>>> reset()
>>> for i in range(20):
        circle(10+5*i,180)
```



Εδώ σχεδιάσαμε 20 ημικύκλια. Το εσωτερικό έχει ακτίνα 10 pixels και κάθε επόμενο έχει ακτίνα 5 pixels μεγαλύτερη από ότι το προηγούμενο.

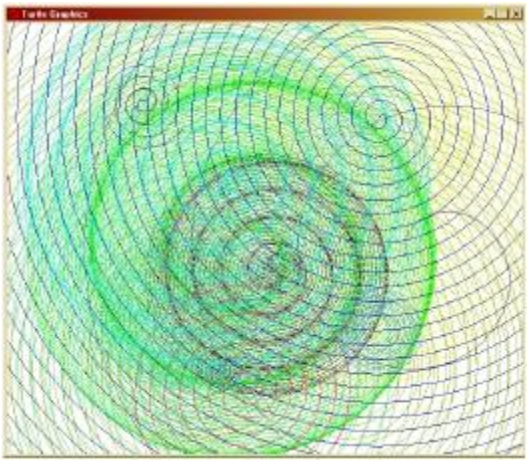
Αν θέλουμε να παίξουμε με τις σπείρες, βολεύει να φτιάξουμε μια συνάρτηση:

```
>>> def spiral(x,y,arc,times,inner,increment):
        up()
        goto(x,y)
```

```
        down()
        for i in range(times):circle(inner+increment*i,arc)
```

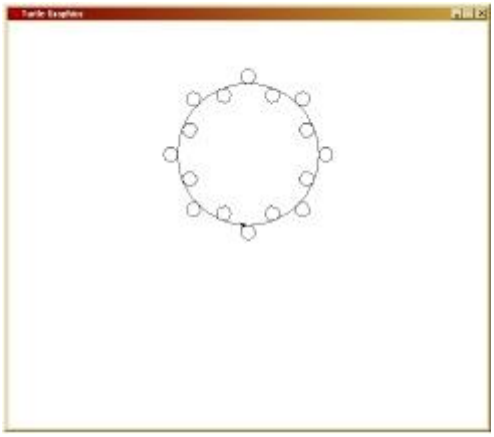
Η συνάρτηση spiral() σχεδιάζει μια σπείρα ξεκινώντας από το σημείο (x,y) και φτιάχνοντας times το πλήθος τόξα, καθένα γωνίας arc μοιρών. Το πρώτο θα έχει ακτίνα inner και η ακτίνα κάθε επόμενου θα αυξάνεται κατά increment. Και τώρα μπορούμε να φτιάξουμε π.χ. 20 τυχαίες σπείρες:

```
>>> reset()
>>> speed('fastest')
>>> for i in range(20):
        color(random(),random(),random())
        x=randint(-200,200)
        y=randint(-200,200)
        arc=randint(-360,360)
        spiral(x,y,arc,randint(10,100),randint(-
30,30),randint(-10,10))
```



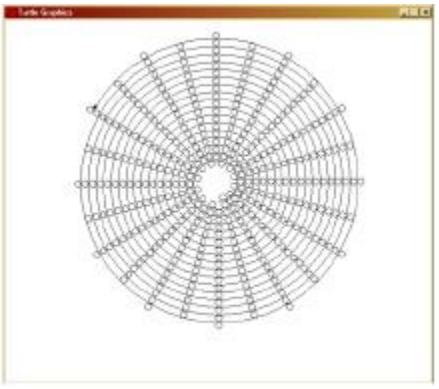
Με τα τόξα και λίγη προσοχή στον προγραμματισμό μπορούμε να φτιάξουμε αρκετά ωραία γραφικά. Για παράδειγμα, προσπαθήστε να καταλάβετε πώς ο παρακάτω κώδικας σχεδιάζει το διπλανό σχήμα:

```
>>> reset()
>>> for i in range(8):
        circle(100,22.5)
        circle(10)
        circle(100,22.5)
        circle(-10)
```



Και μετά συνδυάστε τον με μια σπείρα όπως:

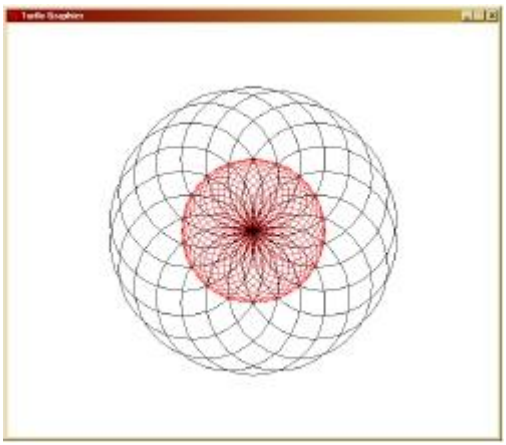
```
>>> reset()
>>> for i in range(200):
        circle(30+i,15)
        circle(5)
        circle(30+i,15)
        circle(-5)
```



Δείτε επίσης έναν άλλον σχετικά απλό τρόπο να φτιάχνουμε εντυπωσιακά σχήματα, χρησιμοποιώντας μόνο κύκλους:

```
>>> reset()
>>> speed('fastest')
>>> color('red')
>>> for i in range(36):
        circle(50)
        left(10)

>>> color('black')
>>> for i in range(18):
        circle(100)
        left(20)
```



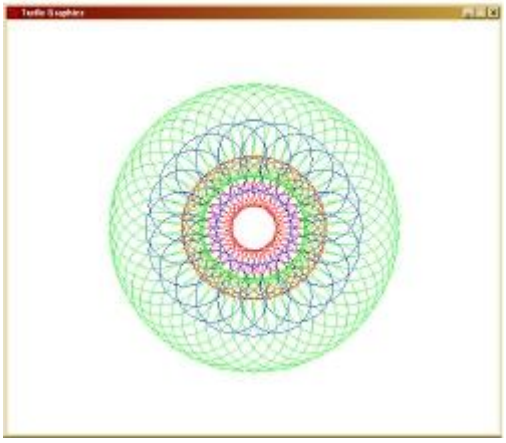
Και πάλι, μπορούμε να εξελίξουμε τον αλγόριθμο σε συνάρτηση που θα φτιάχνει «μαργαρίτες» με τις παραμέτρους που θα δίνουμε:

```
>>> def daisy(inner, outer, number, colour):
        angle=360.0/number
        radius=-(outer-inner)/2.
        color(colour)
        up()
        goto(0,0)
        right(90)
        forward(inner)
        left(90)
        for i in range(number):
                down()
                circle(radius)
                up()
                circle(inner,angle)
```

Η ιδέα είναι η εξής: δυο ομόκεντροι κύκλοι με ακτίνες inner και outer ορίζουν ένα «δακτυλίδι» μέσα στο οποίο η συνάρτηση σχεδιάζει number το πλήθος ισαπέχοντες κύκλους χρώματος colour.

Και τώρα μπορούμε να γράψουμε, πχ.:

```
>>> reset()
>>> speed('fastest')
>>> daisy(30,100,36,'red')
>>> daisy(50,150,20,'blue')
>>> daisy(70,200,50,'green')
```



5.7 Ασκήσεις

Αφού μελετήσετε τα παραδείγματα των ενοτήτων που προηγήθηκαν, εκπονήστε τις παρακάτω ασκήσεις. Αναρτήστε τα αρχεία σας ως ένα ενιαίο συμπιεσμένο αρχείο. Δώστε ιδιαίτερη προσοχή στην πληρότητα και την εμφάνιση του κώδικά σας.

Άσκηση #1

Επισκεφτείτε την ιστοσελίδα του Project Euler (http://projecteuler.net).

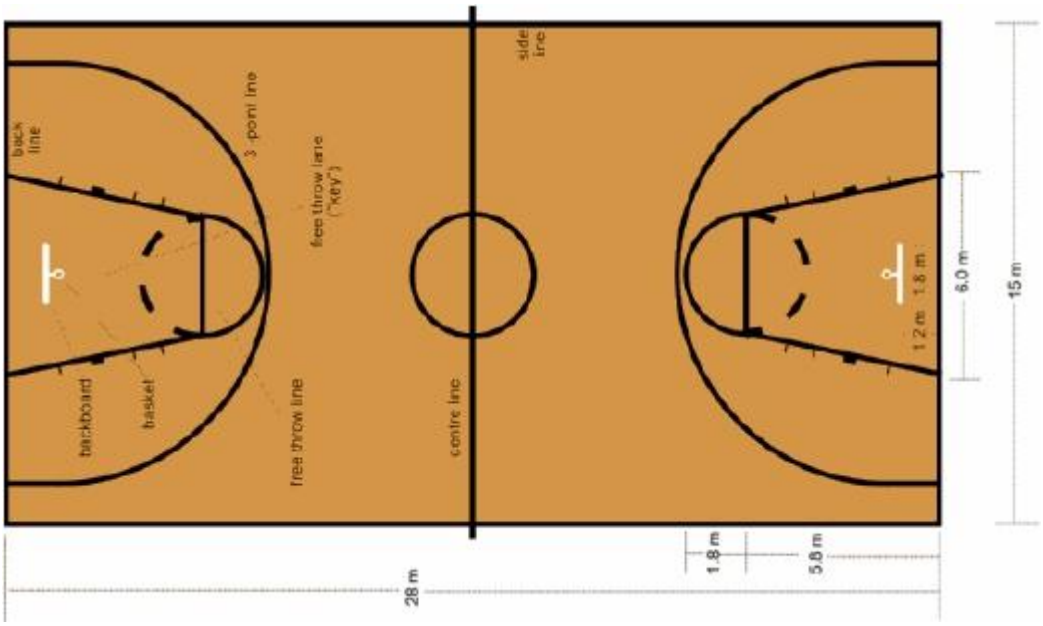
Επιλέξτε δύο προβλήματα[3] από εκεί και λύστε τα με τη βοήθεια της Python. Χρησιμοποιήστε ξεχωριστό αρχείο κώδικα για κάθε πρόβλημα και μέσα σε αυτό γράψτε με σχόλια τον αριθμό του προβλήματος, την εκφώνησή του (copy-paste από το site) κα το αποτέλεσμα[4] που βρήκατε.

Άσκηση #2

Χρησιμοποιήστε τη βιβλιοθήκη turtle της Python για να σχεδιάσετε την κάτοψη ενός γηπέδου ποδοσφαίρου ή μπάσκετ. Δεν μας ενδιαφέρουν οι ακριβείς αναλογίες, ούτε τα χρώματα, αλλά προσπαθήστε να πετύχετε σωστά τα σχήματα (παραλληλόγραμμα, ημικύκλια, κλπ). Μαζί με τον κώδικα δώστε και την εικόνα (screen capture) του γηπέδου όπως το σχεδίασε το πρόγραμμά σας.

[3] Τα προβλήματα με μικρό αύξοντα αριθμό είναι τα πιο εύκολα.
[4] Μπορείτε αν θέλετε να ελέγξετε αν η απάντηση που βρήκατε είναι σωστή, μέσω του ίδιου site.


90-120m
16.5m
Half-way line
Penalty area
18-yard box
5.5m
40.3m
Centre circle
9.15m
Centre spot
Penalty spot
Goal
7.32m
x
2.44m
11m
5.5m
11m
45-90m
9 m


back line
3 point line
free throw lane ("key")
side line
backboard
basket
free throw line
centre line
1.2 m
1.8 m
6.0 m
15 m
1.8 m
5.8 m
28 m

Σημειώματα

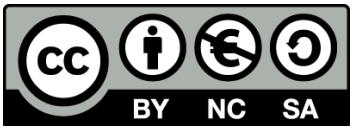
Σημείωμα Ιστορικού Εκδόσεων Έργου

Το παρόν έργο αποτελεί την έκδοση 1.0.

- Έκδοση 1.0 διαθέσιμη εδώ.

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης



Διατήρηση Σημειωμάτων

- Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Σημείωμα Χρήσης Έργων Τρίτων

Το Έργο αυτό κάνει χρήση των ακόλουθων έργων:

Εικόνες/Σχήματα/Διαγράμματα/Φωτογραφίες

Εικόνες: Προέρχονται από Python IDLE.

Πίνακες

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.