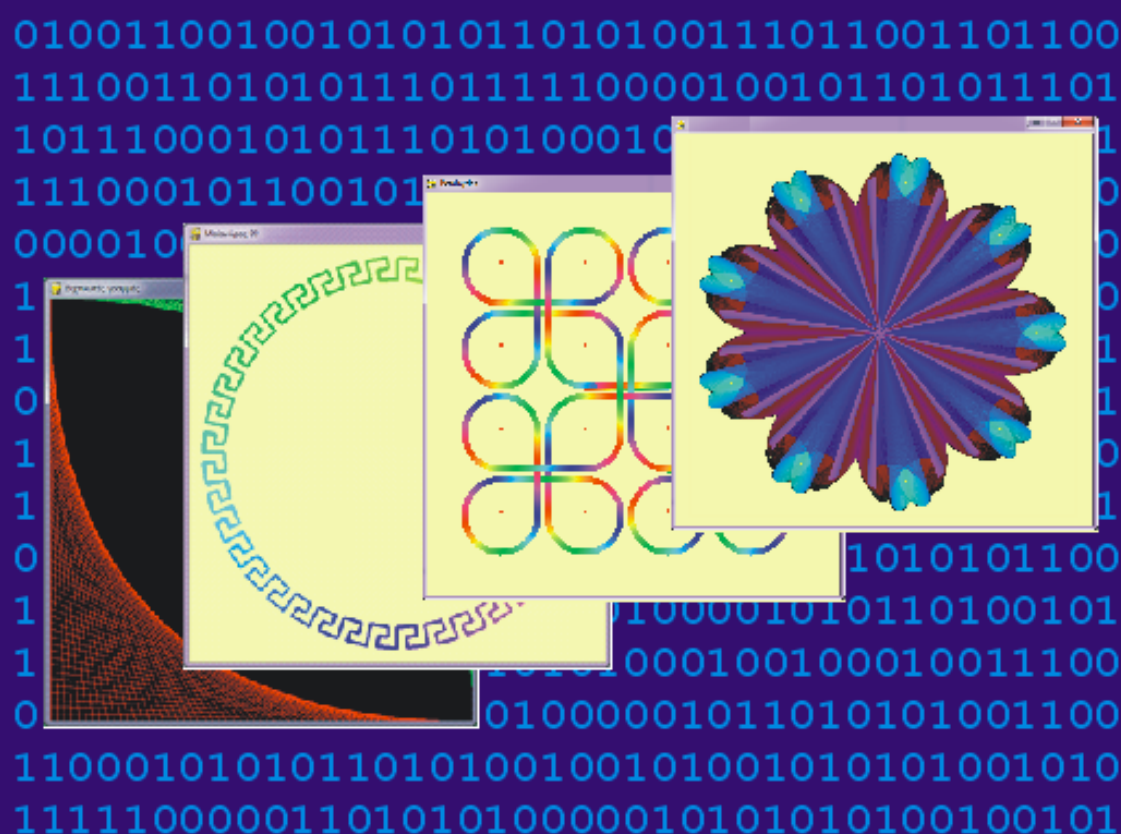


ΓΙΩΡΓΟΣ Ι. ΨΑΡΑΚΗΣ

ΚΑΛΛΙΤΕΧΝΙΕΣ ΜΕ ΥΠΟΛΟΓΙΣΤΗ

Βήμα βήμα προς
μικρές και μεγάλες καλλιτεχνικές δημιουργίες
με τη βοήθεια της εύκολης
γλώσσας προγραμματισμού Python



ΣΥΝΙΣΤΑΤΑΙ ΚΑΙ ΓΙΑ ΑΡΧΑΡΙΟΥΣ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ

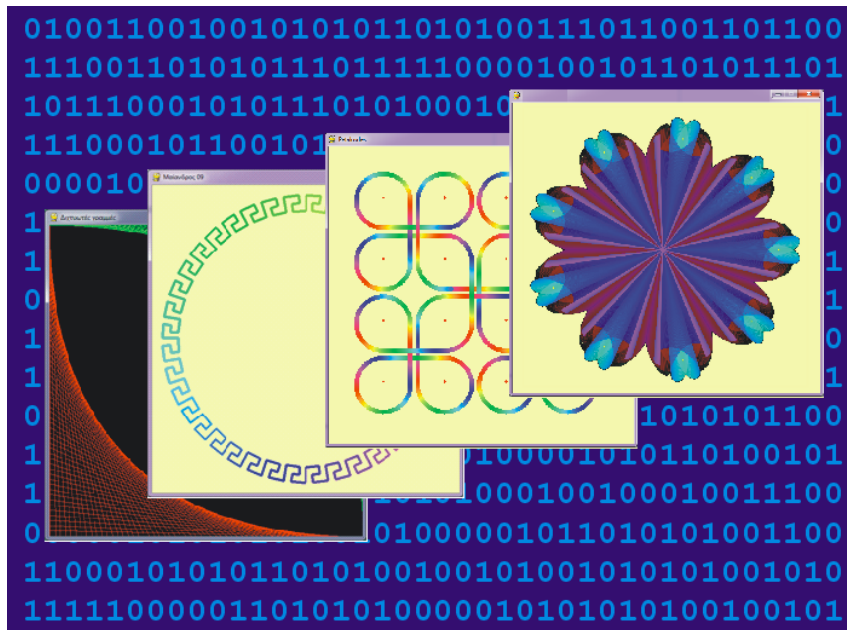


e-book

ΓΙΩΡΓΟΣ Ι. ΨΑΡΑΚΗΣ

ΚΑΛΛΙΤΕΧΝΙΕΣ ΜΕ ΥΠΟΛΟΓΙΣΤΗ

Βήμα βήμα
προς μικρές και μεγάλες καλλιτεχνικές δημιουργίες
με τη βοήθεια της εύκολης γλώσσας προγραμματισμού Python.



Τίτλος βιβλίου: ΚΑΛΛΙΤΕΧΝΙΕΣ ΜΕ ΥΠΟΛΟΓΙΣΤΗ
Συγγραφέας: ΓΙΩΡΓΟΣ Ι. ΨΑΡΑΚΗΣ
Ιστοσελίδα βιβλίου: www.7777777.gr
ISBN 978-960-88910-4-3
Copyright © 2016: ΓΙΩΡΓΟΣ Ι. ΨΑΡΑΚΗΣ

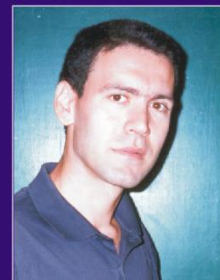
Άλλα βιβλία του ίδιου συγγραφέα:

1. ΘΗΣΑΥΡΟΣ ΠΡΟΒΛΗΜΑΤΩΝ, ΛΙΒΑΝΗ 2003, ISBN 978-960-14-0449-3
2. ΜΙΚΡΟΙ ΕΠΑΝΑΣΤΑΤΕΣ, ΨΑΡΑΚΙ 2005, ISBN 978-960-88910-0-0
3. ΘΗΣΑΥΡΟΣ ΓΡΙΦΩΝ, ΨΑΡΑΚΙ 2007, ISBN 978-960-88910-1-2
4. ΠΡΟΓΡΑΜΜΑΤΙΖΟΝΤΑΣ ΕΝΑ ΣΚΑΚΙ, ΨΑΡΑΚΙ 2014, ISBN 978-960-88910-2-9



ΔΗΜΙΟΥΡΓΙΕΣ ΕΝΤΥΠΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΗΣ ΜΟΡΦΗΣ

ΧΑΝΙΑ 2016



Ο Γιώργος Ψαράκης κατάγεται από τα Χανιά της Κρήτης. Έχει κάνει βασικές σπουδές στην Πληροφορική στο Πανεπιστήμιο Κρήτης και μεταπτυχιακές σπουδές στο τμήμα Γραφικών Τεχνών και Πολυμέσων του Ανοικτού Πανεπιστημίου Πατρών. Εργάζεται ως καθηγητής στη Β/θμια Εκπαίδευση και παράλληλα ασχολείται με τη συγγραφή βιβλίων. Το παρόν είναι το πέμπτο βιβλίο του.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ	3
ΚΕΦΑΛΑΙΟ 0, Εγκατάσταση του περιβάλλοντος εργασίας.....	4
ΚΕΦΑΛΑΙΟ 1, Το πρώτο μας πρόγραμμα	5
ΚΕΦΑΛΑΙΟ 2, Γραμμές κύκλοι, τετράγωνα, χρώματα, συντεταγμένες.....	8
ΚΕΦΑΛΑΙΟ 3, Πολλές γραμμές	11
ΚΕΦΑΛΑΙΟ 4, Διχτυωτές γραμμές	12
ΚΕΦΑΛΑΙΟ 5, Ακτινωτό σχήμα (1)	14
ΚΕΦΑΛΑΙΟ 6, Ακτινωτό σχήμα (2)	15
ΚΕΦΑΛΑΙΟ 7, Ακτινωτό σχήμα (3)	15
ΚΕΦΑΛΑΙΟ 8, Ακτινωτό σχήμα (4)	16
ΚΕΦΑΛΑΙΟ 9, Ακτινωτό σχήμα (5)	17
ΚΕΦΑΛΑΙΟ 10, Ακτινωτό σχήμα (6)	18
ΚΕΦΑΛΑΙΟ 11, Ακτινωτό σχήμα (7)	18
ΚΕΦΑΛΑΙΟ 12, Αστεροειδές	18
ΚΕΦΑΛΑΙΟ 13, Διχτυωτό αστεροειδές	19
ΚΕΦΑΛΑΙΟ 14, Διχτυωτό αστεροειδές δίχρωμο	20
ΚΕΦΑΛΑΙΟ 15, Η εντολή <code>pygame.draw.aaline</code>	20
ΚΕΦΑΛΑΙΟ 16, Μαϊάνδρος 01	21
ΚΕΦΑΛΑΙΟ 17, Μαϊάνδρος 02	21
ΚΕΦΑΛΑΙΟ 18, Μαϊάνδρος 03	21
ΚΕΦΑΛΑΙΟ 19, Μαϊάνδρος 04	22
ΚΕΦΑΛΑΙΟ 20, Μαϊάνδρος 05	22
ΚΕΦΑΛΑΙΟ 21, Μαϊάνδρος 06	23
ΚΕΦΑΛΑΙΟ 22, Μαϊάνδρος 07	23
ΚΕΦΑΛΑΙΟ 23, Μαϊάνδρος 08	26
ΚΕΦΑΛΑΙΟ 24, Μαϊάνδρος 09	27
ΚΕΦΑΛΑΙΟ 25, Μαϊάνδρος 10	28
ΚΕΦΑΛΑΙΟ 26, Έγχρωμος κύκλος 01	28
ΚΕΦΑΛΑΙΟ 27, Έγχρωμος κύκλος 02	29
ΚΕΦΑΛΑΙΟ 28, Έγχρωμος κύκλος 03	31
ΚΕΦΑΛΑΙΟ 29, Έγχρωμος κύκλος 04	32
ΚΕΦΑΛΑΙΟ 30, Έγχρωμος κύκλος 05 με εντολή <code>import</code>	33
ΚΕΦΑΛΑΙΟ 31, Ευθύγραμμες κινήσεις με διανύσματα	34
ΚΕΦΑΛΑΙΟ 32, Καμπύλες κινήσεις με διανύσματα	36
ΚΕΦΑΛΑΙΟ 33, Κίνηση πλανήτη γύρω από ήλιο	38
ΚΕΦΑΛΑΙΟ 34, Εφέ κομήτη	42
ΚΕΦΑΛΑΙΟ 35, Πράσινος κομήτης.....	44
ΚΕΦΑΛΑΙΟ 36, Πολλές τροχιές	46
ΚΕΦΑΛΑΙΟ 37, Συμμετρικές τροχιές.....	50
ΚΕΦΑΛΑΙΟ 38, Ζεύγη τροχιών	55
ΚΕΦΑΛΑΙΟ 39, Δημιουργικότητα	60
ΚΕΦΑΛΑΙΟ 40, Σύνθεση.....	63
ΚΕΦΑΛΑΙΟ 41, Πεταλούδες.....	64
ΕΠΙΛΟΓΟΣ	64

ΕΙΣΑΓΩΓΗ

Ο σκοπός του βιβλίου είναι να αναδειχθούν οι καλλιτεχνικές δυνατότητες των υπολογιστών μέσω του προγραμματισμού και να μεταδοθεί στους ενδιαφερόμενους ένας γρήγορος και απλός τρόπος για να εκκινήσουν ίσως τις δικές τους καλλιτεχνικές προσπάθειες.

Είναι γνωστό βέβαια, ότι αρχικά οι υπολογιστές χειρίστηκαν αριθμούς και κείμενα. Όμως κάποια στιγμή έγινε αντιληπτό ότι μπορούν να χειριστούν και να εμφανίσουν σχήματα ή και εικόνες στις οθόνες. Ήταν αναπόφευκτο να προχωρήσουν οι εξελίξεις και προς αυτό που ονομάζεται διεθνώς Computer Art.

Προσωπικά, στο Πανεπιστήμιο Κρήτης πριν πολλά χρόνια, μπόρεσα για πρώτη φορά να εκφραστώ με προγραμματιστικές εντολές σημείων και γραμμών, δημιουργώντας απλές καλλιτεχνίες. Από τότε μέχρι σήμερα έχουν αλλάξει κυριολεκτικά τα πάντα στον τομέα! Το μοναδικό πράσινο χρώμα στις μαύρες οθόνες έχει δώσει τη θέση του σε εκατομμύρια χρώματα. Οι γλώσσες προγραμματισμού απέκτησαν ισχυρές σχεδιαστικές εντολές, έχουν δημιουργηθεί πανίσχυρα προγράμματα επεξεργασίας εικόνας όπως το Photoshop και βέβαια έχει εξελιχθεί πάρα πολύ ο τομέας της επεξεργασίας βίντεο και κινούμενης εικόνας.

Σε αυτό το βιβλίο θα γίνει προσπάθεια να μεταδοθούν απλά, συστηματικά και γρήγορα στον αναγνώστη οι ισχυρές σχεδιαστικές εντολές του περιβάλλοντος Python/Pygame μέσω αρκετών παραδειγμάτων. Ακολουθήθηκε εν πολλοίς το παράδειγμα του βιβλίου μου «ΠΡΟΓΡΑΜΜΑΤΙΖΟΝΤΑΣ ΕΝΑ ΣΚΑΚΙ», από το οποίο θα ήταν χρήσιμο να υπάρχει η γνώση των βασικών κεφαλαίων του. Πολλές σχεδιαστικές ιδέες του βιβλίου αυτού δεν εξαρτώνται από τη συγκεκριμένη γλώσσα προγραμματισμού Python και μπορεί κανείς να τις μεταφέρει και σε άλλες γλώσσες και περιβάλλοντα. Αν μπορέσει να παρακινηθεί ο αναγνώστης προς τις δικές του πρωτότυπες καλλιτεχνικές δημιουργίες, θα έχει επιτευχθεί και ο κύριος σκοπός του βιβλίου.

Ο συγγραφέας

Γιώργος Ψαράκης

Το βιβλίο αφιερώνεται στην Εικαστική Τέχνη

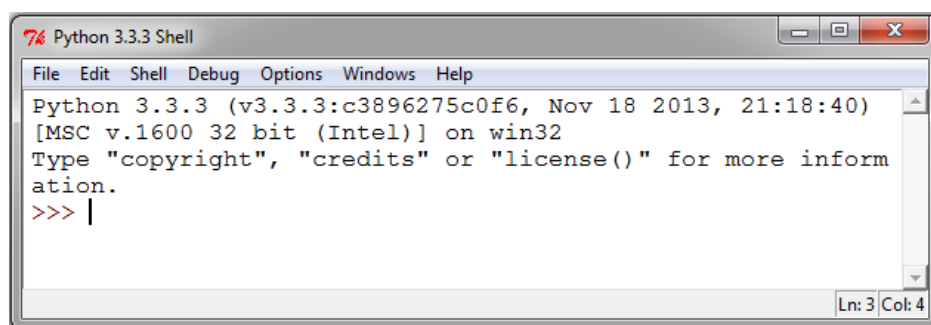
ΚΕΦΑΛΑΙΟ 0, Εγκατάσταση του περιβάλλοντος εργασίας

Μόνο δύο αρχεία είναι απαραίτητα να κατεβούν από το Διαδίκτυο, ώστε να μπορεί να ξεκινήσει κανείς να μελετά με αυτό το βιβλίο. Πρόκειται πρώτον για το αρχείο εγκατάστασης της γλώσσας προγραμματισμού Python σε περιβάλλον Windows και δεύτερον για το αρχείο επέκτασης των δυνατοτήτων της γλώσσας Python, το λεγόμενο Pygame.

Είναι απαραίτητο οι εκδόσεις Python και Pygame να είναι συνεργάσιμες μεταξύ τους, συμβατές όπως λέγεται. Στο βιβλίο γίνεται χρήση Python 3.3 και Pygame 1.9 των οποίων τα αντίστοιχα αρχεία μπορούν να κατεβούν από την ιστοσελίδα www.7777777.gr. Μπορούν επίσης να αναζητηθούν και στις ιστοσελίδες python.org, pygame.org, bitbucket.org/pygame/pygame/downloads ή και 1lyk-kissam.chan.sch.gr/downloads.

Μετά το κατέβασμα (download) των δύο αρχείων, εγκαθιστούμε πρώτα την Python και θα δημιουργηθεί ο φάκελος C:/Python33. Εγκαθιστούμε στη συνέχεια το Pygame, προσέχοντας να δηλώσουμε τον ίδιο φάκελο εγκατάστασης, δηλαδή C:/Python33 αντί για C:/PythonX που προτείνει η εγκατάσταση.

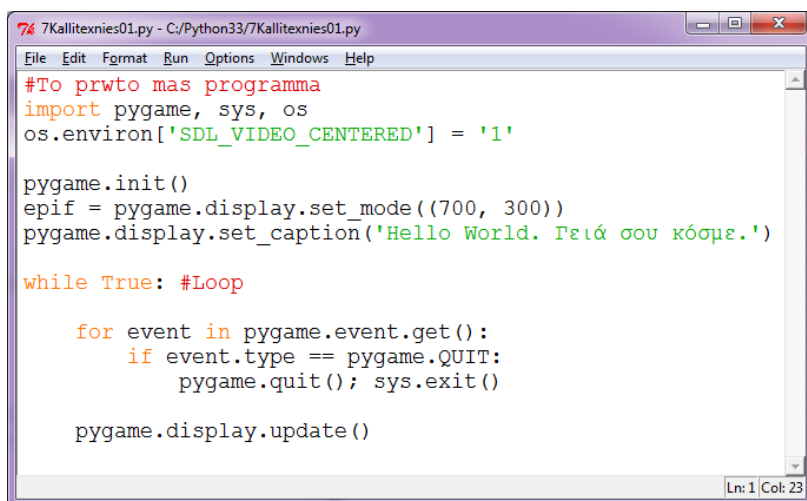
Αν γίνουν όλα σωστά, στο μενού Έναρξη/Όλα τα προγράμματα θα έχει δημιουργηθεί η επιλογή φακέλου Python 3.3. Με κλικ σε αυτόν εμφανίζεται υπομενού, και μας ενδιαφέρει πρωτίστως η επιλογή IDLE (Python GUI) με την οποία θα ανοίγει το περιβάλλον εργασίας μας. Με κλικ σε αυτήν, ανοίγει το λεγόμενο κέλυφος της Python (π.χ. Python 3.3.3 Shell).



Δίπλα στο σύμβολο προτροπής >>>, ο κατακόρυφος κέρσορας που αναβοσβήνει περιμένει τις εντολές μας. Π.χ. γράφοντας 3+4 και πατώντας Enter, θα εμφανιστεί το αποτέλεσμα 7. Όμως, δεν θα μας απασχολήσουν προς το παρόν οι εντολές κελύφους, ζήτημα που θα εξεταστεί σε επόμενα κεφάλαια κάθε φορά που θα κριθεί σκόπιμο. Μας ενδιαφέρει περισσότερο να έχουμε γρήγορα αποτελέσματα σε γραφικά. Γι' αυτό το σκοπό, από το μενού File/New window..., θα ανοίγουμε ένα δεύτερο παράθυρο, μέσα στο οποίο θα γράφουμε τα προγράμματά μας.

ΚΕΦΑΛΑΙΟ 1, Το πρώτο μας πρόγραμμα

Πληκτρολογούμε τις παρακάτω εντολές, όχι στο κέλυφος αλλά σε νέο παράθυρο της Python που ανοίγουμε από File/New window...



```
7Kallitexnies01.py - C:/Python33/7Kallitexnies01.py
File Edit Format Run Options Windows Help
#Το prwto mas programma
import pygame, sys, os
os.environ['SDL_VIDEO_CENTERED'] = '1'

pygame.init()
epif = pygame.display.set_mode((700, 300))
pygame.display.set_caption('Hello World. Γειά σου κόσμε.')

while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()

    pygame.display.update()
```

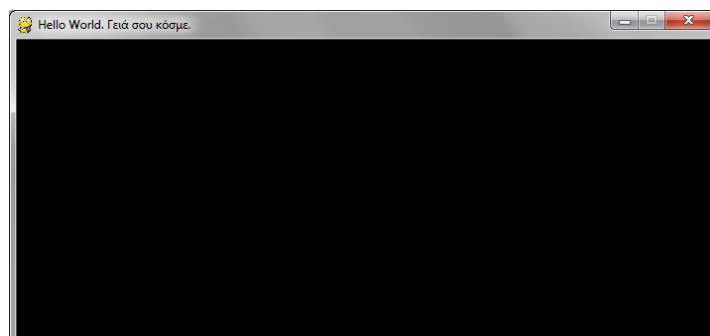
Να δοθεί προσοχή, ώστε στις γραμμές που ξεκινούν πιο δεξιά όπως η γραμμή με την εντολή for, η εσοχή να περιλαμβάνει ακριβώς τέσσερις κενούς χαρακτήρες (spacebar). Αντί για τέσσερα κενά συνιστάται να πατά κανείς το πλήκτρο Tab. Παρόμοια, η γραμμή με την εντολή if περιλαμβάνει αρχικά δύο Tab, ή αλλιώς οκτώ κενούς χαρακτήρες κτλ. Στην γλώσσα προγραμματισμού Python, αυτό το ζήτημα των Tab και των εσοχών είναι κομβικής σημασίας, πρέπει να τηρείται με ακρίβεια και θα εξηγηθεί περισσότερο αργότερα.



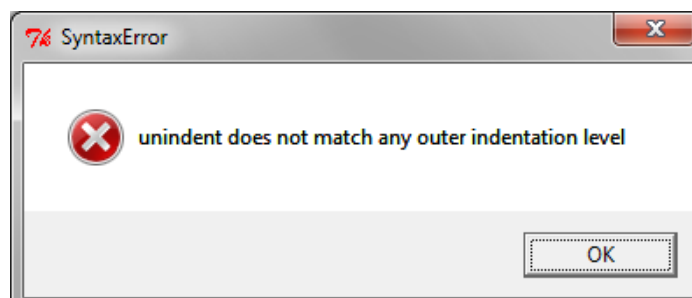
7Kallitexnies01.py

Αποθηκεύουμε το πρώτο πρόγραμμά μας με διαδικασία παρόμοια με αυτήν της αποθήκευσης π.χ. ενός αρχείου του κειμενογράφου Word. Από το οριζόντιο μενού επιλογών, επιλέγουμε File/Save... Στο παράθυρο αποθήκευσης που ανοίγει, πρέπει να δοθούν τρία στοιχεία: το όνομα αρχείου, ο τύπος του και η θέση στο σκληρό δίσκο που επιθυμούμε να αποθηκευτεί. Ως όνομα αρχείου δίνουμε π.χ. 7Kallitexnies01.py. Ο τύπος αρχείου ορίζεται αυτόματα και είναι Python File, χαρακτηρίζεται από την επέκταση .py και από το διπλανό σύμβολο. Ως θέση αποθήκευσης ορίζουμε το φάκελο εγκατάστασης της Python, δηλαδή C:/Python33.

Και τώρα ήρθε η ώρα να εκτελέσουμε το πρώτο μας πρόγραμμα! Να το τρέξουμε όπως λέγεται. Από το οριζόντιο μενού επιλέγουμε Run/Run Module... ή ισοδύναμα πατάμε F5. Αν δούμε να ανοίγει ένα νέο παράθυρο με μαύρη οθόνη όπως παρακάτω, τότε όλα πληκτρολογήθηκαν καλά. Η εκτέλεση προκάλεσε να ανοιχθεί και τρίτο παράθυρο, στο οποίο παρουσιάζεται η εκτέλεση του προγράμματος.



Αντιθέτως, αν δούμε στην οθόνη μας μήνυμα σαν το παρακάτω, τότε πατάμε OK και επανελέγχουμε την ορθότητα των γραμμών κώδικα που πληκτρολογήσαμε.



Όπως ήδη προαναφέρθηκε, απαιτείται απόλυτη ακρίβεια στις εσοχές!

Ας εξετάσουμε μία μία τις εντολές του κώδικα. Πρώτα απ' όλα, παρατηρούμε τις κόκκινες εντολές, οι οποίες όλες ξεκινούν με τον χαρακτήρα της δίσησης #. Όταν σε μια γραμμή κώδικα παρουσιαστεί ο χαρακτήρας #, κάθε τι μετά από αυτό αποτελεί σχόλιο του προγραμματιστή για το συγκεκριμένο τμήμα εντολών. Έχει αποδειχθεί χρήσιμο να σχολιάζουμε σύντομα την λειτουργία των εντολών μας. Βεβαίως, τα σχόλια δεν αποτελούν εντολή προς εκτέλεση από τον υπολογιστή.

Η αρχική εντολή `import` εισάγει/συνδέει κάποια χρήσιμα προγραμματιστικά στοιχεία από τα τμήματα (modules όπως λέγονται στα αγγλικά) `pygame`, `sys` και `os` της γλώσσας προγραμματισμού Python. Θα την χρησιμοποιούμε αρχικά στα τυφλά σε όλα τα προγράμματα μας και με τον καιρό θα εμβαθύνουμε. Η επόμενη εντολή `os.environ['SDL_VIDEO_CENTERED'] = '1'` καθορίζει ότι το παράθυρο εξόδου θα εμφανιστεί στο κέντρο της οθόνης. Θα χρησιμοποιείται και σε όλα τα επόμενα προγράμματά μας. Δεν θα μπορούσε να εκτελεστεί εάν προηγουμένως δεν είχαμε εισάγει το τμήμα (module) `os`.

Η εντολή `pygame.init()` καθιστά έτοιμο το πρόγραμμά μας να εκτελεί λειτουργίες του τμήματος `pygame`, λειτουργίες που αφορούν ισχυρές δυνατότητες γραφικών, ήχου κ.α.

Η εντολή `pygame.display.set_mode((700, 300))` δημιουργεί το παράθυρο εκτέλεσης. Στις παρενθέσεις παρατηρούμε το ζευγάρι αριθμών (700, 300). Αυτό καθορίζει το πλάτος και το ύψος του παραθύρου εκτέλεσης σε εικονοστοιχεία (pixel). Αριστερά του συμβόλου = χρησιμοποιούμε μια μεταβλητή όπως λέγεται, συγκεκριμένα τη μεταβλητή `erif`, η οποία αντιπροσωπεύει το παράθυρο εκτέλεσης. Ή αλλιώς την επιφάνεια σχεδίασης που δημιουργήθηκε στο δεξί σκέλος. Θα μπορούσε να απουσιάζει αυτή η μεταβλητή και ο χαρακτήρας = και θα αρκούσε να γράψουμε `pygame.display.set_mode((700, 300))`. Όμως σε επόμενα παραδείγματα όπου πρέπει να γίνουν σχεδιαστικές επεμβάσεις στην αρχική επιφάνεια, θα φανεί ιδιαίτερα χρήσιμο να αναφερόμαστε σε αυτήν με κάποιο όνομα. Στην περίπτωση μας επιλέξαμε το όνομα `erif`, ως αρχικά λατινικά γράμματα της λέξης επιφάνεια, όμως θα μπορούσαμε να επιλέξουμε οποιοδήποτε άλλο όνομα με λατινικά γράμματα.

Με την εντολή `pygame.display.set_caption('Hello World.Γεια σου κόσμο.')` δηλώνουμε τι θέλουμε να εμφανίζεται στον τίτλο του παραθύρου εκτέλεσης. Παρατηρούμε ότι μέσα στις παρενθέσεις της εντολής, πρέπει να δηλωθεί ένα όρισμα, συγκεκριμένα ένα

κείμενο, το οποίο μπορεί να είναι και μικτό αγγλοελληνικό. Τα κείμενα οριοθετούνται με αποστρόφους ('), και στο πληκτρολόγιο βρίσκονται συνήθως αριστερά του πλήκτρου Enter.

Στις προηγούμενες τρεις εντολές υπάρχει κάτι κοινό. Όλες εκκινούν με το πρόθεμα **pygame** ακολουθούμενο από τελεία. Αυτό απλά σημαίνει ότι όλες ανήκουν στις εντολές του πακέτου **pygame**. Παρατηρούμε ότι και οι τρεις εντολές έχουν στο τέλος τους παρενθέσεις μέσα στις οποίες μπαίνουν τα κατάλληλα ορίσματα, ή αλλιώς οι κατάλληλες παράμετροι. Ειδικά η εντολή **init** δεν παίρνει παραμέτρους. Η εντολή **display.set_mode** παίρνει ως όρισμα ένα ζευγάρι τιμών, στην περίπτωση μας το ζευγάρι (700,300) και γι αυτό το λόγο παρατηρούνται διπλές παρενθέσεις. Η εντολή **display.set_caption** παίρνει ως όρισμα μέσα στις παρενθέσεις ένα κείμενο σε αποστρόφους.

Η εντολή **while True**: ακολουθείται πάντα από ένα μπλοκ εντολών γραμμένων σε εσοχή, και δημιουργεί μια αστραπιαία και αέναη εκτέλεση των εντολών αυτών (έως και χιλιάδες φορές το δευτερόλεπτο). Όλες αυτές οι εντολές πρέπει να γραφούν σε εσοχή και αφορούν την αλληλεπίδραση δεδομένων ποντικιού, πληκτρολογίου και αντίστοιχης ανταπόκρισης στην οθόνη, δηλαδή στο παράθυρο εκτέλεσης. Ας εξετάσουμε την παρακάτω εντολή.

```
for event in pygame.event.get():  
    if event.type == pygame.QUIT:  
        pygame.quit(); sys.exit()
```

Η εντολή **for event...** θα ελέγχει συνεχώς για συμβάντα ποντικιού ή πληκτρολογίου. Εδώ, με την ακόλουθη **if event.type == pygame.QUIT**:, ελέγχεται τουλάχιστον ενέργεια τερματισμού του παραθύρου εκτέλεσης, αλλά αργότερα θα ελέγχονται πιθανές άλλες ενέργειες του χρήστη, όπως κλικ ποντικιού ή πληκτρολόγηση. Αργότερα θα εξηγηθεί το τμήμα αυτό περισσότερο.

Στις εντολές σε εσοχή της **while True**:, θα υπάρχει σχεδόν πάντα και η εντολή **pygame.display.update()**, η οποία στέλνει στην οθόνη τα επίκαιρα δεδομένα που σχεδιάστηκαν. Ειδικά σε αυτό το πρώτο μας πρόγραμμα, το οποίο είναι στατικό και όπου δηλαδή δεν υπάρχει αλληλεπίδραση, μπορεί και να απουσιάζει.

Όλες οι εντολές σε εσοχή της εντολής **while**, περιγράφονται και ως βασικό Loop, και όπως θα δούμε στη συνέχεια, αυτές θα είναι και οι πρωταγωνιστικές. Οτιδήποτε πριν την εντολή **while**, θα εκτελεστεί μόνο μια φορά. Αντίθετα, οι εντολές μετά την **while**, θα εκτελούνται ξανά και ξανά.

Κλείνουμε το παράθυρο εκτέλεσης από το κόκκινο κουμπί πάνω δεξιά. Έχουμε στο μυαλό ότι αυτό το πρώτο πρόγραμμα, θα χρησιμεύσει ως ο σκελετός για το επόμενο. Με μικρές αλλαγές θα δημιουργηθεί το δεύτερο πρόγραμμά μας. Παρόμοια, από το δεύτερο πρόγραμμα με μικρές αλλαγές θα δημιουργηθεί το τρίτο κτλ. Καλό είναι, λοιπόν, με την επιλογή File/Save As... να δημιουργούμε ακριβή αντίγραφα από τα προγράμματά μας, να τα τροποποιούμε κατάλληλα και να προχωράμε. Εξάλλου, έτσι μπορούμε να δημιουργούμε αντίγραφα και να πειραματιζόμαστε άφοβα σε αυτά. Για παράδειγμα, αποθηκεύουμε το προηγούμενο πρόγραμμα και ως **test.py**, αλλάζουμε το κείμενο του τίτλου με την εντολή **pygame.display.set_caption** και το τρέχουμε! Ή αλλάζουμε π.χ. τις διαστάσεις ύψους και πλάτους του παραθύρου εκτέλεσης. Ή αλλάζουμε π.χ. την ονομασία της μεταβλητής **epif** με μια άλλη.

Πρέπει να προειδοποιηθεί ότι θα συναντήσουμε αρκετές αγγλικές λέξεις καθώς θα μελετάμε αυτό το βιβλίο. Προϋποτίθεται λοιπόν, όχι μόνο να γνωρίζουμε τα αγγλικά γράμματα, αλλά να γνωρίζουμε και μερικές λέξεις όπως for, if, exit, quit, get, κάτι που θα βοηθήσει αρκετά στην κατανόηση του κώδικα. Γενικότερα, όλες οι γλώσσες προγραμματισμού χρησιμοποιούν τέτοιες λέξεις στη σύνταξή τους, ώστε ο κώδικας να γίνεται πιο προσιτός, λιγότερο αφηρημένος και κατά συνέπεια ευκολότερος στην εκμάθηση, στη διόρθωση και στη μελλοντική του επέκταση. Στο περιβάλλον που θα εργαστούμε, μερικές τέτοιες λέξεις έχουν πορτοκαλί χρωματισμό, διότι ανήκουν στις λεγόμενες λέξεις κλειδιά (keywords) της γλώσσας Python.

Και κάτι ακόμα. Αν θεωρεί κάποιος προχωρημένος αναγνώστης ότι χάνει πολύτιμο χρόνο πληκτρολογώντας τα προγράμματα του βιβλίου, μπορεί και να τα κατεβάσει από την ιστοσελίδα www.7777777.gr. Όμως αυτό δεν συνιστάται καθόλου στους νέους προγραμματιστές, οι οποίοι όσο περισσότερο κώδικα πληκτρολογήσουν, τόσο περισσότερο και θα ωφεληθούν, ακόμα και από τα λάθη τους στην πληκτρολόγηση! Ίσως σε κάποιον δεν τρέχει το πρόγραμμα και δεν μπορεί να εντοπίσει τα λάθη που έχει κάνει στην πληκτρολόγηση, οπότε πάλι είναι χρήσιμο να προστρέξει στο έτοιμο και να το συγκρίνει με το δικό του. Το κυριότερο πάντως είναι, να κατανοούμε καλά κάθε πρόγραμμα που θα μελετάμε και να εμβαθύνουμε στους κανόνες ορθής γραφής και σύνταξης των εντολών.

ΚΕΦΑΛΑΙΟ 2, Γραμμές κύκλοι, τετράγωνα, χρώματα, συντεταγμένες

Ήρθε η ώρα να προγραμματίσουμε κάτι πιο ενδιαφέρον. Γράφουμε τον παρακάτω κώδικα σε νέο αρχείο Python, ή τροποποιούμε κατάλληλα το πρώτο πρόγραμμα. Αποθηκεύουμε π.χ. ως 7Kallitexnies02.py.

```
7Kallitexnies02.py - C:\Python33\7Kallitexnies02.py
File Edit Format Run Options Windows Help
import pygame, sys, os
os.environ['SDL_VIDEO_CENTERED'] = '1'

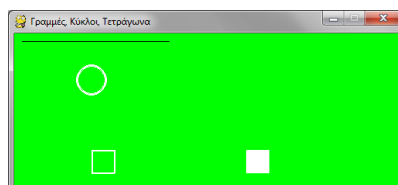
pygame.init()
epif = pygame.display.set_mode((500,200))
pygame.display.set_caption('Γραμμές, Κύκλοι, Τετράγωνα')

#Entoles sxediashs
epif.fill((0,255,0))
pygame.draw.line(epif, (0,0,0), (10,10), (200,10), 1)
pygame.draw.circle(epif, (255,255,255), (100,60), 20, 3)
pygame.draw.rect(epif, (255,255,255), (100,150,30,30), 2)
pygame.draw.rect(epif, (255,255,255), (300,150,30,30), 0)

while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()

    pygame.display.update()
```

Τρέχουμε τον κώδικα είτε με Run/Run Module, είτε με πιο σύντομο τρόπο πατώντας το πλήκτρο συντόμευσης F5. Πρέπει να εμφανιστεί στην οθόνη κάτι σαν το παρακάτω:



Ας δούμε πώς έγινε αυτό. Στις αρχικές εντολές, σε σχέση με το πρώτο πρόγραμμα, παρατηρούμε κάποιες μικρές διαφορές στα ορίσματα. Στην εντολή `pygame.display.set_mode()` ορίσαμε άλλες διαστάσεις πλάτους και ύψους παραθύρου εκτέλεσης, συγκεκριμένα 500 και 200 αντί για 700 και 300 του πρώτου προγράμματός μας. Και στην εντολή `pygame.display.set_caption()` ορίσαμε άλλο κείμενο τίτλου, συγκεκριμένα 'Γραμμές, Κύκλοι, Τετράγωνα'. Στη συνέχεια παρατηρούμε πέντε νέες εντολές:

```
epif.fill((0,255,0))
pygame.draw.line(epif,(0,0,0),(10,10),(200,10),1)
pygame.draw.circle(epif,(255,255,255),(100,60),20,3)
pygame.draw.rect(epif,(255,255,255),(100,150,30,30),2)
pygame.draw.rect(epif,(255,255,255),(300,150,30,30),0)
```

Η πρώτη από αυτές γεμίζει με πράσινο χρώμα την επιφάνεια του παραθύρου εκτέλεσης, η δεύτερη σχεδιάζει μια ευθεία μαύρη γραμμή, η τρίτη έναν λευκό κύκλο, η τέταρτη ένα λευκό τετράγωνο και η πέμπτη ένα γεμισμένο λευκό τετράγωνο.

Πριν εξηγηθούν οι εντολές αυτές, είναι ανάγκη σε αυτό το σημείο να γίνει μια παρένθεση και να κατανοήσουμε καλά δύο ζητήματα: το ένα αφορά τους χρωματισμούς με τους οποίους σχεδιάζουμε, ενώ το άλλο αφορά τις θέσεις στις οποίες σχεδιάζουμε.

ΧΡΩΜΑΤΑ

Αν εξετάσουμε προσεκτικά τα ορίσματα στις παραπάνω εντολές, θα διαπιστώσουμε ότι το ένα από αυτά είναι πάντα μια τριάδα αριθμών π.χ. (0, 255, 0). Αυτή η τριάδα καθορίζει ακριβώς το αναπαριστώμενο χρώμα, ανάλογα με την τιμή που έχουν οι τρεις αριθμοί. Οι τιμές κυμαίνονται από 0 έως και 255. Η πρώτη τιμή καθορίζει τη συμμετοχή του κόκκινου, η δεύτερη καθορίζει τη συμμετοχή του πράσινου και η τρίτη καθορίζει τη συμμετοχή του μπλε στον τελικό χρωματισμό. Έτσι, αντιλαμβανόμαστε ότι η τριάδα (0, 255, 0) σημαίνει πράσινο, ενώ οι τριάδες (0, 0, 0) και (255, 255, 255) σημαίνουν μαύρο και λευκό αντίστοιχα (ως γνωστόν η σύνθεση όλων των χρωμάτων δημιουργεί λευκό ενώ η απουσία όλων δημιουργεί μαύρο). Αυτό το σύστημα αναπαράστασης χρωμάτων, που ονομάζεται και σύστημα RGB από τα αρχικά των λέξεων Red Green Blue), χρησιμοποιείται ευρύτατα στον προγραμματισμό και θα το γνωρίσουμε καλύτερα με την εμπειρία μας. Παρακάτω βλέπουμε τους βασικούς χρωματισμούς και τις αντίστοιχες τριάδες τους.

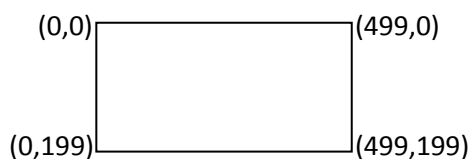
	(255,0,0)
	(0,255,0)
	(0,0,255)
	(255,255,255)
	(0,0,0)
	(177,177,177)

Σημειώνεται ότι πρέπει να δίνεται απόλυτη προσοχή, ώστε η τιμή κάθε χρωματισμού να βρίσκεται πάντα εντός του διαστήματος [0,256), δηλαδή να είναι μεγαλύτερη ή ίση του 0 και μικρότερη του 256. Επιτρέπονται δεκαδικές τιμές στους χρωματισμούς, οι οποίες στρογγυλοποιούνται αυτόματα σε ακέραιες κατά τη σχεδίαση.

ΘΕΣΗ ΣΧΕΔΙΑΣΗΣ

Στην εντολή `pygame.draw.line(epif, (0,0,0), (10,10), (200,10), 1)` που σχεδιάζει τη μαύρη γραμμή, παρατηρούμε πέντε ορίσματα: Με το πρώτο όρισμα, δηλαδή τη μεταβλητή `epif`, δηλώνεται σε ποια επιφάνεια θα σχεδιαστεί η γραμμή, με το δεύτερο όρισμα δηλώνεται το χρώμα, και με το πέμπτο όρισμα δηλώνεται το πάχος της γραμμής σε pixel. Παρατηρούμε ως τρίτο και τέταρτο όρισμα δύο ζευγάρια αριθμών (10,10) και (200,10). Αυτά, υποδηλώνουν συγκεκριμένα σημεία της επιφάνειας. Στην εντολή σχεδίασης γραμμής καθορίζουν με απόλυτη ακρίβεια τα δύο άκρα της γραμμής. Το σημείο (10,10) απέχει 10 pixel από αριστερά και 10 pixel από επάνω, ενώ το σημείο (200,10) απέχει 200 pixel από αριστερά και 10 pixel από επάνω.

Γενικά, κάθε σημείο της επιφάνειας έχει τις δικές του αποστάσεις από αριστερά και από επάνω, οι οποίες ονομάζονται και συντεταγμένες του σημείου. Το επάνω αριστερό σημείο δεν έχει τις φυσιολογικές συντεταγμένες (1,1) που ίσως περίμενε κανείς, αλλά (0,0). Έτσι, στο παράθυρό μας το πιο κάτω δεξί σημείο αναφέρεται με (499,199) και όχι με (500,200). Αυτό γίνεται κατανοητό ίσως, αν προσπαθήσει κανείς να υπολογίσει την απόσταση του πρώτου pixel από τα άκρα. Από την επάνω γραμμή απέχει πράγματι 0, ενώ από την αριστερή πλευρά απέχει επίσης 0. Επίσης, για όσους γνωρίζουν για το επίπεδο καρτεσιανό σύστημα συντεταγμένων στα μαθηματικά, ας παρατηρηθεί η σημαντική διαφορά της αρχής. Στα γραφικά των υπολογιστών έχει επικρατήσει το επάνω αριστερό σημείο ως αρχή, ενώ το επίπεδο καρτεσιανό σύστημα αναπαρίσταται με την αρχή του κάτω αριστερά!



Στην εντολή `pygame.draw.circle(epif, (255,255,255), (100,60), 20, 3)` επίσης παρατηρούμε ένα όρισμα που έχει τη μορφή συντεταγμένων σημείου, συγκεκριμένα το ζευγάρι αριθμών (100, 60). Στις εντολές σχεδίασης κύκλων, με αυτό το τρίτο όρισμα δηλώνεται το κέντρο του κύκλου. Με το τέταρτο όρισμα δηλώνεται το μέγεθος της ακτίνας του κύκλου ενώ με το πέμπτο δηλώνεται το πάχος της γραμμής του κύκλου σε pixel.

Η εντολή `pygame.draw.rect(epif, (255,255,255), (100,150,30,30), 2)` είναι εντολή σχεδίασης τετραγώνου. Το πρώτο όρισμα (η μεταβλητή `epif`) αφορά την επιφάνεια που θα σχεδιαστεί το τετράγωνο. Το δεύτερο όρισμα αφορά τον χρωματισμό, ο οποίος στην περίπτωση μας είναι το λευκό. Το τρίτο όρισμα είναι μια τετράδα αριθμών, η οποία χαρακτηρίζει τη θέση το πλάτος και το ύψος του τετραγώνου. Οι δύο πρώτοι αριθμοί της τετράδας αφορούν την απόσταση του τετραγώνου από αριστερά και από επάνω, με άλλα λόγια τις συντεταγμένες του επάνω αριστερού σημείου. Οι δύο επόμενοι αριθμοί της τετράδας αφορούν το πλάτος και το ύψος του σχεδιαζόμενου τετραγώνου. Το τελευταίο όρισμα αφορά το πάχος γραμμής σε pixel του τετραγώνου. Όταν αυτό το όρισμα είναι 0 όπως συμβαίνει με την τελευταία εντολή σχεδίασης, το τετράγωνο σχεδιάζεται γεμισμένο και αυτό ισχύει και για τη σχεδίαση κύκλων.

Ας σημειωθεί ότι επιτρέπονται αναφορές σε σημεία εκτός της επιφάνειας σχεδίασης! Για παράδειγμα, αν δοκιμάσουμε να σχεδιάσουμε γραμμή με άκρα τα σημεία (10,10) και

(800,10), τότε αυτή θα σχεδιαστεί, παρόλο που το δεύτερο σημείο βρίσκεται εκτός καμβά. Οι αναφορές σε σημεία πάνω ή αριστερά από τον καμβά, γίνονται με αρνητικούς αριθμούς.

Ας σημειωθεί επίσης ότι υπάρχουν και άλλες εντολές σχεδίασης όπως π.χ. `pygame.draw.ellipse(epif, (0,0,255), (200,100,40,80), 2)` ή `pygame.draw.polygon(epif, (0,0,255), ((4,70), (10,100), (35,105), (45,95)), 3)`, οι οποίες σχεδιάζουν ελλείψεις και πολύγωνα αντίστοιχα. Στην έλλειψη πρέπει να δοθεί ως όρισμα και μια τετράδα, δηλαδή ένα νοητό τετράγωνο, μέσα στο οποίο θέλουμε να εμφανιστεί (όπως στην εντολή τετραγώνων `rect`). Στο πολύγωνο πρέπει συν τοις άλλοις να δοθούν τα ζεύγη συντεταγμένων των κορυφών του. Αν επιθυμεί κανείς να σχεδιάσει σημείο, τότε το πετυχαίνει αυτό με την εντολή γραμμής, ορίζοντας ως αρχή και τέλος τις συντεταγμένες του ίδιου αυτού σημείου.

Συνιστάται ιδιαίτερα να δημιουργήσουμε αντίγραφο του παραπάνω προγράμματος και να δοκιμάσουμε διάφορες τιμές ορισμάτων στις εντολές. Κάθε κεφάλαιο, πρέπει να χωνεύεται πολύ καλά και ο καλύτερος τρόπος γι' αυτό, είναι ο συνεχής πειραματισμός και η αυτενέργεια. Πειραματιζόμαστε λοιπόν με κύκλους και γραμμές, με χρώματα και με θέσεις και φτιάχνουμε τις δικές μας δημιουργίες.

ΚΕΦΑΛΑΙΟ 3, Πολλές γραμμές

Γράφουμε τον παρακάτω κώδικα σε νέο αρχείο Python, είτε τροποποιούμε κατάλληλα το προηγούμενο πρόγραμμα. Αποθηκεύουμε π.χ. ως 7Kallitexnies03.py.

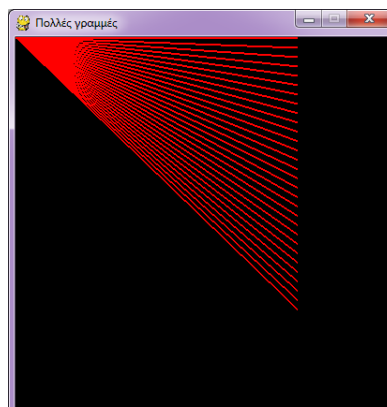
```
7Kallitexnies03.py - C:/Python33/7Kallitexnies03.py
File Edit Format Run Options Windows Help
import pygame, sys, os
os.environ['SDL_VIDEO_CENTERED'] = '1'

pygame.init()
epif = pygame.display.set_mode((400,400))
pygame.display.set_caption('Πολλές γραμμές')

for i in range(30):
    pygame.draw.line(epif, (255,0,0), (0,0), (300,10*i), 2)

while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.display.update()
```

Τρέχοντας το πρόγραμμα, παρατηρούμε το εξής αποτέλεσμα:



Ας εξετάσουμε τον κώδικα στο σημείο της εντολής `for`, όπου και έγιναν οι μόνες αλλαγές.

Γενικά, η εντολή `for` έχει ως αποτέλεσμα την πολλαπλή εκτέλεση των εντολών που ακολουθούν σε εσοχή. Η έκφραση δίπλα στο `for` καθορίζει το πλήθος των επαναλήψεων. Στην περίπτωσή μας, η έκφραση `i in range(30)` έχει ως αποτέλεσμα η μεταβλητή `i` να πάρει διαδοχικά τις τιμές 0, 1, 2, ... 29. Η εντολή `range(...)` επιστρέφει μια λίστα με ακέραιους, τόσους όσο το όρισμα. Για κάποιους λόγους, που δεν θα εξετάσουμε εδώ, οι ακέραιοι που επιστρέφονται γενικά από την `range(n)` δεν είναι 1, 2, 3, ... n , αλλά 0, 1, ... $n-1$. Δηλαδή περιλαμβάνεται το 0 ενώ δεν περιλαμβάνεται ο τελευταίος όρος. Όταν τα ορίσματα της `range` είναι δύο, π.χ. `range(1, 5)`, επιστρέφονται οι ακέραιοι από το πρώτο όρισμα συμπεριλαμβανόμενο, έως το τελευταίο, μη συμπεριλαμβανόμενο. Στο παράδειγμα θα επιστραφούν 1, 2, 3, 4. Καλό είναι να πειραματιστεί κανείς στο κέλυφος της Python με την `range`. Γράφοντας `list(range(8))` δίπλα στο σύμβολο προτροπής `>>>` και πατώντας Enter, θα εμφανιστεί αμέσως η λίστα με τους οκτώ αντίστοιχους αριθμούς από το 0 έως το 7. Γράφοντας `list(range(1,10))` θα εμφανιστούν οι αριθμοί από 1 έως 9. Είναι χρήσιμο να γνωρίζει κανείς επίσης, ότι αν δοθούν τρία ορίσματα στην `range(...)`, τότε επιστρέφεται λίστα αριθμών με βηματισμό σύμφωνα με το τρίτο όρισμα! Π.χ. η `range(2,102,2)` επιστρέφει τους ζυγούς 2, 4, 6, ... 100. Πληκτρολογώντας στο κέλυφος `list(range(2,102,2))` και πατώντας Enter, το επιβεβαιώνουμε αυτό.

Η εντολή `pygame.draw.line` σε εσοχή που ακολουθεί την `for`, θα εκτελεστεί λοιπόν τριάντα φορές. Παρατηρούμε ότι κάθε φορά θα σχεδιάζεται μια κόκκινη γραμμή. Όμως το όρισμα που καθορίζει το δεύτερο άκρο της γραμμής που θα σχεδιάζεται κάθε φορά, ίσως φαίνεται ακατανόητο. Είναι το ζεύγος $(300, 10*i)$. Ενώ, λοιπόν, η περιοχή σχεδίασης `erif`, το χρώμα σχεδίασης $(255, 0, 0)$, το πρώτο άκρο $(0, 0)$, καθώς και το πλάτος (2) είναι σαφώς καθορισμένα, δεν συμβαίνει το ίδιο με το δεύτερο άκρο της γραμμής.

Πρέπει να γίνει κατανοητό ότι σε κάθε επανάληψη θα προκύπτει διαφορετικό ζεύγος συντεταγμένων για το δεύτερο άκρο, δηλαδή διαφορετικό σημείο που θα καταλήγει η γραμμή που θα ξεκινά πάντα από το σημείο $(0, 0)$. Στην πρώτη επανάληψη της `for` που το `i` έχει τιμή 0, το σημείο κατάληξης θα είναι το $(300, 0)$. Στην δεύτερη επανάληψη που το `i` έχει τιμή 1, το σημείο κατάληξης θα είναι το $(300, 10)$. Στην τρίτη επανάληψη που το `i` έχει τιμή 2, το σημείο θα είναι το $(300, 20)$ κτλ. Στην τριακοστή επανάληψη που το `i` έχει τιμή 29, το σημείο θα είναι το $(300, 290)$. Όλα αυτά τα ζεύγη ή αλλιώς σημεία, θα χρησιμοποιηθούν επαναληπτικά στην εντολή `pygame.draw.line`, έχοντας ως αποτέλεσμα τη σχεδίαση τριάντα διαφορετικών γραμμών.

Ας δοκιμάσει κανείς να αλλάξει και το πρώτο άκρο της γραμμής χρησιμοποιώντας έκφραση που να περιέχει τη μεταβλητή `i`, π.χ. $(i*10, 0)$ αντί για $(0, 0)$. Θα εκπλαγεί με τα αποτελέσματα αλλά και θα χωνέψει περισσότερο αυτό το κεφάλαιο.

ΚΕΦΑΛΑΙΟ 4, Διχτυωτές γραμμές

Γράφουμε τον παρακάτω κώδικα, ο οποίος λίγο διαφέρει από αυτόν του προηγούμενου κεφαλαίου.

```
7Kallitexnies04.py - C:/Python33/7Kallitexnies04.py
File Edit Format Run Options Windows Help
import pygame, sys, os
os.environ['SDL_VIDEO_CENTERED'] = '1'

SIZE=600 #Megethos parathyroy

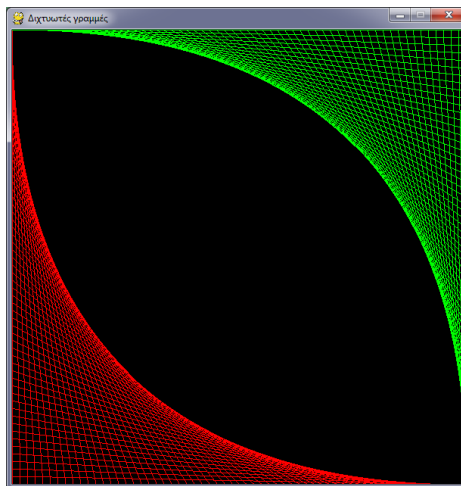
pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Διχτυωτές γραμμές')

for i in range(0, SIZE, 10):
    pygame.draw.line(epif, (255,0,0), (0,i), (i,SIZE),1)

for i in range(0, SIZE, 10):
    pygame.draw.line(epif, (0, 255, 0), (i,0), (SIZE,i),1)

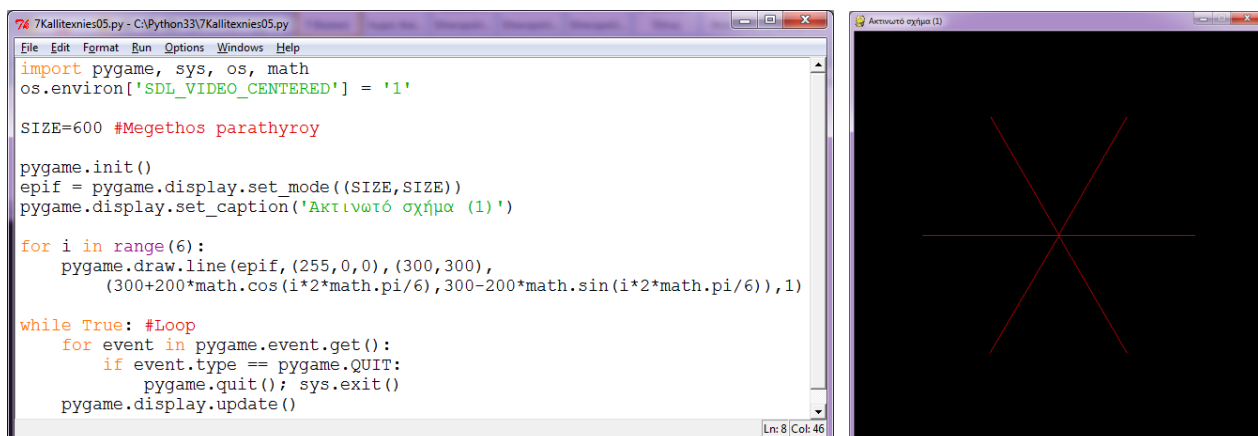
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.display.update()
```

Αποθηκεύουμε π.χ. ως 7Kallitexnies04.py και το τρέχουμε.



Δεν χρειάζονται πολλές επεξηγήσεις καθώς πρόκειται για φυσιολογική συνέχεια του προηγούμενου κεφαλαίου, όπου έχουν γίνει οι σχετικές διευκρινήσεις. Σε ένα σημείο όμως πρέπει κανείς να σταθεί, συγκεκριμένα στην εντολή **SIZE=600**. Με αυτήν, δημιουργείται μια μεταβλητή ονόματος SIZE με αρχική τιμή-περιεχόμενο 600. Όπου στη συνέχεια του κώδικα συναντά κανείς τη μεταβλητή SIZE, είναι υποχρεωμένος να θεωρήσει εκεί τη τιμή 600. Κατά σύμβαση, οι μεταβλητές που γράφονται εξ' ολοκλήρου με κεφαλαία γράμματα, θεωρούνται σταθερές μεταβλητές, δηλαδή κατά πάσα πιθανότητα δεν πρόκειται να αλλαχθεί η τιμή τους στη συνέχεια. Αυτό γίνεται για λόγους διάκρισης με τις μεταβλητές που θα τροποποιείται η τιμή τους και οι οποίες συνηθίζεται να γράφονται με μικρά γράμματα. Ας δοκιμάσει κανείς να αλλάξει τη τιμή SIZE και να δει τα αποτελέσματα. Ουσιαστικά πρόκειται για το μήκος του τετράγωνου παραθύρου, όμως παίζει καθοριστικό ρόλο στη συνέχεια και στο πλήθος των σχεδιαζόμενων γραμμών στις εντολές **for**.

ΚΕΦΑΛΑΙΟ 5, Ακτινωτό σχήμα (1)



Μεταφέρουμε τον παραπάνω κώδικα σε νέο αρχείο, τον αποθηκεύουμε π.χ. ως 7Kallitexnies05.py και τον τρέχουμε. Εμφανίζεται ένα ακτινωτό.

Παρατηρώντας κανείς τον κώδικα, αντιλαμβάνεται ότι σχεδιάζονται έξι κόκκινες γραμμές που εκκινούν όλες από το κέντρο, δηλαδή από το σημείο (300,300). Αυτό που ίσως δεν είναι άμεσα αντιληπτό είναι πώς επετεύχθη οι έξι αυτές γραμμές να τερματίζουν με τάξη σχηματίζοντας ένα τέλειο εξαπλό ακτινωτό. Σημειώνεται ότι επιτρέπεται να σπάσει μια εντολή σε δεύτερη γραμμή αρκεί το σπάσιμο να γίνει σε σημείο που υπάρχει κόμμα που διαχωρίζει ορίσματα.

Ας δώσουμε τις εξηγήσεις, οι οποίες απαιτούν στοιχειώδη γνώση τριγωνομετρίας. Το σημείο τερματισμού των έξι γραμμών, περιγράφεται ως $(300+200*\text{math.cos}(i*2*\text{math.pi}/6), 300-200*\text{math.sin}(i*2*\text{math.pi}/6))$.

Καθώς η μεταβλητή i θα πάρει διαδοχικά τις τιμές 0, 1, 2, 3, 4, 5, η παράσταση αυτή υπολογίζει διαδοχικά τα σημεία τερματισμού των ακτινών. Γνωρίζουμε ότι μεταξύ των ακτινών δημιουργείται γωνία 60 μοιρών. Επειδή η μονάδα μέτρησης γωνιών όταν χρησιμοποιούνται οι τριγωνομετρικές συναρτήσεις ημιτόνου και συνημιτόνου (math.sin και math.cos) στην Python είναι τα ακτίνια (rad), οι 60 μοίρες πρέπει να γραφτούν ως $2\pi/6$, δηλαδή προγραμματιστικά ως $2*\text{math.pi}/6$. Το πρόθεμα math που χρησιμοποιείται και στον αριθμό π αλλά και στις συναρτήσεις ημιτόνου και συνημιτόνου $\sin$ και $\cos$, υποδηλώνει ότι γίνεται χρήση ενός ειδικού μαθηματικού εργαλείου της Python. Για να μπορεί να γίνει χρήση αυτού του εργαλείου (module), στην πρώτη γραμμή το θέσαμε ως όρισμα της `import`.

```
|| import pygame, sys, os, math ||
```

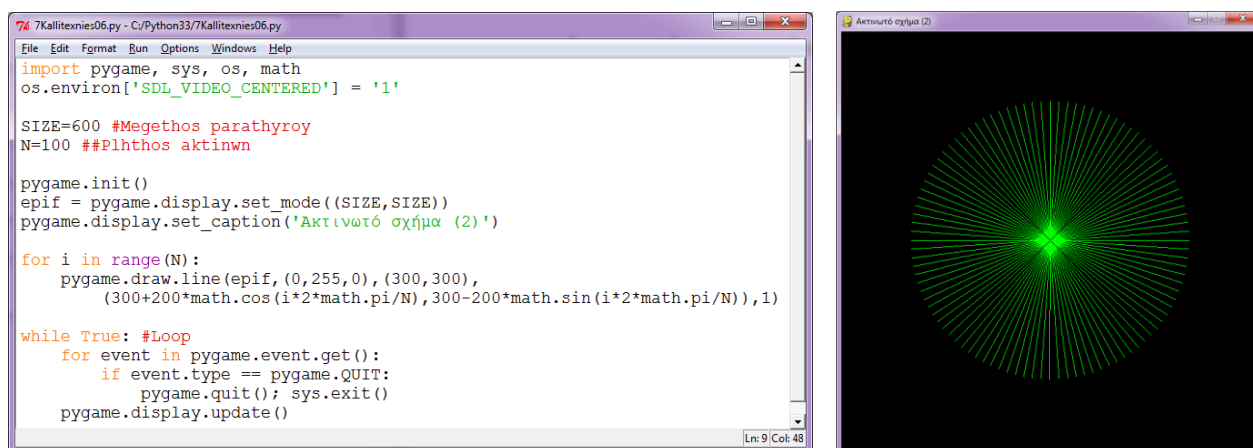
Τα σημεία τερματισμού, βρίσκονται προφανώς πάνω σε κύκλο. Από την τριγωνομετρία, γνωρίζουμε ότι αν σχηματίζεται γωνία 60 μοιρών, τότε το αντίστοιχο σημείο του μοναδιαίου κύκλου έχει συντεταγμένες $\sin(2\pi/6)$ και $\cos(2\pi/6)$. Παρόμοια, αν σχηματίζεται γωνία 120 μοιρών, τότε το αντίστοιχο σημείο του μοναδιαίου κύκλου έχει συντεταγμένες $\sin(2*2\pi/6)$ και $\cos(2*2\pi/6)$, αν σχηματίζεται γωνία 180 μοιρών, τότε το αντίστοιχο σημείο του κύκλου έχει συντεταγμένες $\sin(3*2\pi/6)$ και $\cos(3*2\pi/6)$ κτλ. Αν δεν πρόκειται για μοναδιαίο κύκλο (κύκλο με ακτίνα 1), αλλά για κύκλο ακτίνας 200 όπως στην περίπτωση μας, τότε για να βρεθούν οι συντεταγμένες επάνω στον κύκλο, πρέπει να πολλαπλασιαστούν τα ημίτονα και τα συνημίτονα με την ακτίνα του κύκλου, στην περίπτωση μας με το 200. Έτσι εξηγούνται οι εκφράσεις

$200 * \text{math.cos}(i * 2 * \text{math.pi} / 6)$ και $200 * \text{math.sin}(i * 2 * \text{math.pi} / 6)$. Η πρώτη έκφραση πρέπει να προστεθεί στο 300 καθορίζοντας την οριζόντια θέση του σημείου του κύκλου ενώ η δεύτερη πρέπει να αφαιρεθεί από το 300 καθορίζοντας την κάθετη θέση του σημείου του κύκλου, θέτοντας ως σύμβαση ότι οι ακτίνες σχεδιάζονται με τη σειρά ωριαίου δείκτη ρολογιού 3:00, 1:00, 11:00, 9:00, 7:00, 5:00.

Το παράδειγμα αυτό αλλά και το επόμενο που θα ακολουθήσει δείχνει περίτρανα την αξία της τριγωνομετρίας, την οποία ίσως «αντιπαθήσαμε» λίγο στα σχολικά θρανία, ίσως επειδή δεν κατανοήσαμε τότε την πρακτική της χρησιμότητα.

ΚΕΦΑΛΑΙΟ 6, Ακτινωτό σχήμα (2)

Πώς μπορούμε να τροποποιήσουμε κατάλληλα το προηγούμενο παράδειγμα, ώστε να σχεδιάζεται ακτινωτό περισσότερων ακτίνων; Αυτό περιγράφεται στον κώδικα που ακολουθεί τον οποίο γράφουμε και αποθηκεύουμε π.χ. ως 7Kallitexnies06.py.



Η μόνη ουσιαστική αλλαγή που έχει γίνει σε σχέση με το προηγούμενο πρόγραμμα, είναι οι τρεις αντικαταστάσεις του αριθμού 6 με τη μεταβλητή-σταθερά N. Δηλαδή πλέον, με την αρχική εντολή εκχώρησης $N=100$ καθορίζεται το πλήθος των ακτίνων και στη συνέχεια η for εκτελείται N φορές σχεδιάζοντας N ακτίνες. Όμως τώρα, η αρχική γωνία βάσης δεν θα είναι 60 μοίρες (ή αλλιώς $2\pi/6$ rad), αλλά θα υπολογίζεται διαιρώντας το 2π με το N, με άλλα λόγια ο κύκλος θα γίνεται N κομμάτια ($2\pi/N$ rad). Αυτή η γωνία βάσης, πολλαπλασιαζόμενη διαδοχικά με το i, δηλαδή με 0, 1, 2, 3,...N, θα δίνει τις γωνίες που είναι κάθε φορά απαραίτητες για το σχεδιασμό της αντίστοιχης ακτίνας.

Είναι χρήσιμο, να δοκιμάσουμε πολλές φορές το πρόγραμμα αυτό αλλάζοντας κάποιες τιμές π.χ. το 100 ή ακόμα και το 200 (η ακτίνα του σχεδίου μας) που εμφανίζεται να πολλαπλασιάζει τις τριγωνομετρικές συναρτήσεις. Ειδικά η χρήση του γενικού N προσδίδει δύναμη στο πρόγραμμά μας, καθώς μια μόνο αλλαγή στη τιμή του προκαλεί εντυπωσιακή αλλαγή αποτελέσματος.

ΚΕΦΑΛΑΙΟ 7, Ακτινωτό σχήμα (3)

Αν η εκκίνηση των ακτίνων δεν γίνεται από το κεντρικό σημείο (300,300) αλλά τα σημεία εκκίνησης των ακτίνων βρίσκονται και αυτά πάνω σε κύκλο, τότε προκύπτουν επίσης ενδιαφέρουσες δημιουργίες. Γράφουμε τον κώδικα που ακολουθεί και τον αποθηκεύουμε π.χ. ως 7Kallitexnies07.py.

```

74 7Kallitexnies07.py - C:\Python33\7Kallitexnies07.py
File Edit Format Run Options Windows Help
import pygame, sys, os, math
os.environ['SDL_VIDEO_CENTERED'] = '1'

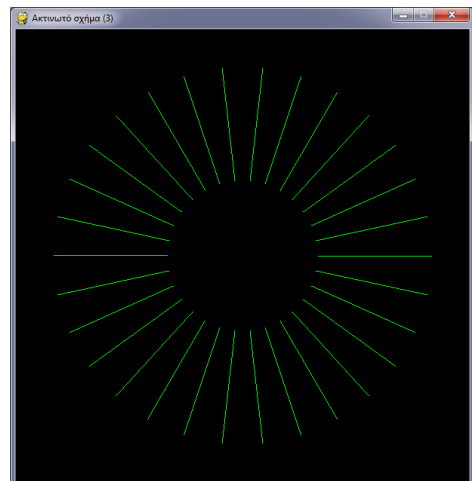
SIZE=600 #Megethos parathyroy
X1=300; Y1=300; R1=100 #Idiotites prwtoy kykloy
X2=300; Y2=300; R2=250 #Idiotites deyteroy kykloy
N=30 #Plhthos aktinwn

pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Ακτινωτό σχήμα (3)')

for i in range(N):
    pygame.draw.line(epif, (0,255,0),
        (X1+R1*math.cos(i*2*math.pi/N), Y1-R1*math.sin(i*2*math.pi/N)),
        (X2+R2*math.cos(i*2*math.pi/N), Y2-R2*math.sin(i*2*math.pi/N)), 1)

while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.display.update()

```



Παρατηρεί κανείς ότι τα σημεία εκκίνησης των ακτίνων βρίσκονται σε κύκλο με κέντρο $(X1,Y1)$ και ακτίνα $R1$, ενώ τα σημεία τερματισμού των ακτίνων βρίσκονται σε κύκλο με κέντρο $(X2,Y2)$ και ακτίνα $R2$. Η χρήση όλων αυτών των μεταβλητών-σταθερών προσδίδει μια δημιουργική γενικότητα στο πρόγραμμα. Π.χ. αλλάζοντας μόνο τη τιμή 100 του $R1$ στην εντολή εκχώρησης $R1=100$, θα αλλάξει ανάλογα το αποτέλεσμα, δηλαδή τα σημεία εκκίνησης θα βρεθούν πάνω σε άλλο κύκλο.

ΚΕΦΑΛΑΙΟ 8, Ακτινωτό σχήμα (4)

Αν δεν ενώνονται παρεμφερή σημεία των δύο κύκλων, αλλά ενώνονται με κάποια διαφορά φάσης (γωνίας), τότε προκύπτουν επίσης ενδιαφέρουσες δημιουργίες. Γράφουμε τον κώδικα που ακολουθεί και τον αποθηκεύουμε π.χ. ως 7Kallitexnies08.py.

```

74 7Kallitexnies08.py - C:\Python33\7Kallitexnies08.py
File Edit Format Run Options Windows Help
import pygame, sys, os, math
os.environ['SDL_VIDEO_CENTERED'] = '1'

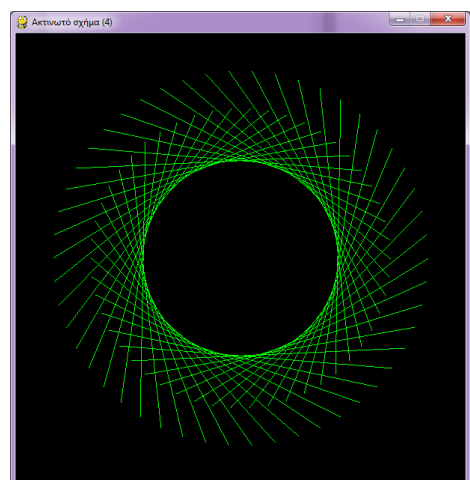
SIZE=600 #Megethos parathyroy
X1=300; Y1=300; R1=200 #Idiotites prwtoy kykloy
X2=300; Y2=300; R2=250 #Idiotites deyteroy kykloy
N=50 #Plhthos aktinwn
D=15 #Diafora fashs

pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Ακτινωτό σχήμα (4)')

for i in range(N):
    pygame.draw.line(epif, (0,255,0),
        (X1+R1*math.cos(i*2*math.pi/N), Y1-R1*math.sin(i*2*math.pi/N)),
        (X2+R2*math.cos((i+D)*2*math.pi/N), Y2-R2*math.sin((i+D)*2*math.pi/N)), 1)

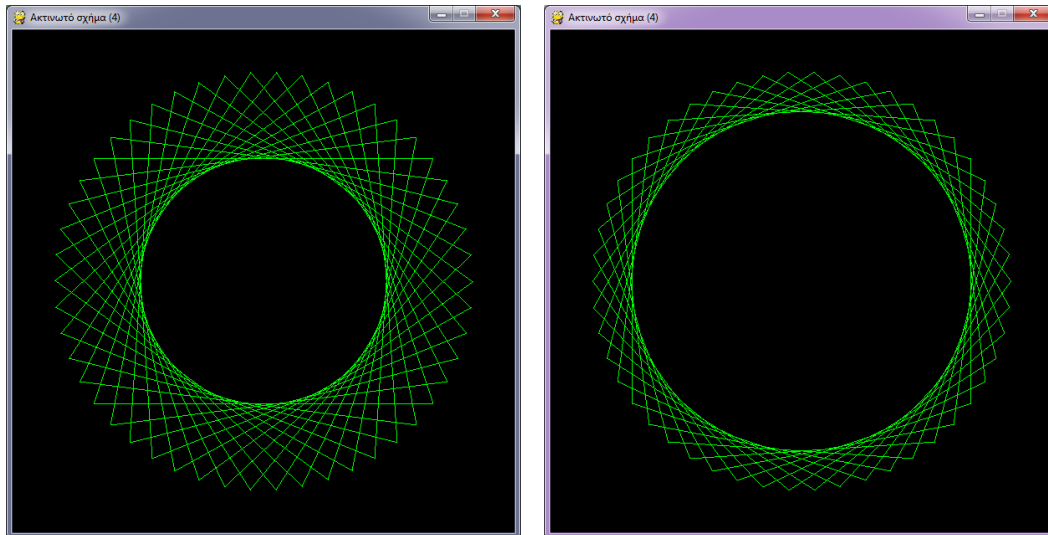
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.display.update()

```



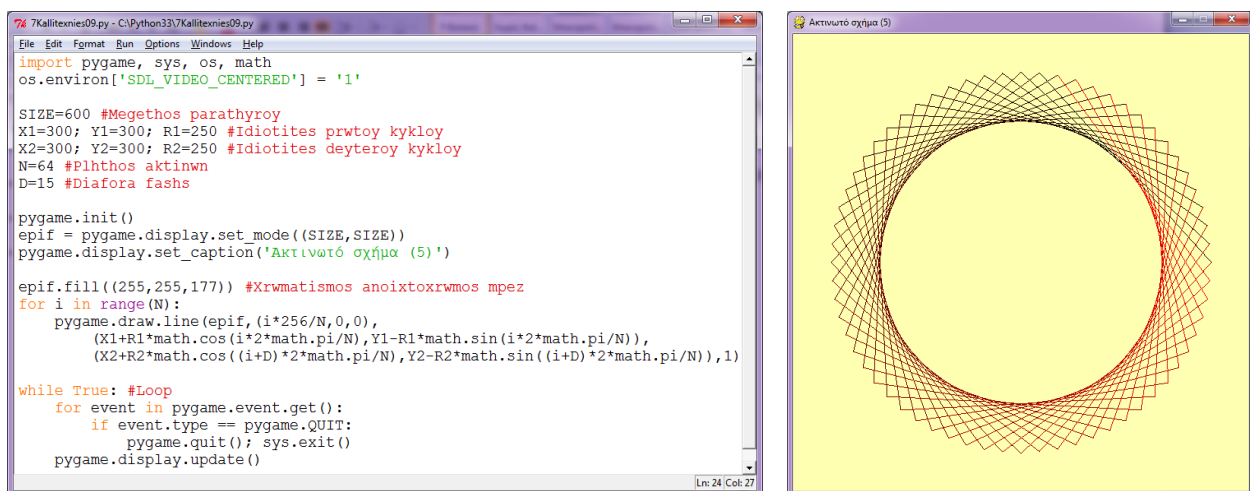
Παρατηρεί κανείς ότι στο σημείο τερματισμού, η βασική γωνία $2\pi/N$ δεν πολλαπλασιάζεται πλέον με i αλλά με $i+D$. Έτσι πετυχαίνονται διαφορετικές γωνίες στα σημεία τερματισμού (σε σχέση με τις γωνίες στα σημεία εκκίνησης), ανάλογα με την τιμή της διαφοράς φάσης D .

Αν οι δύο κύκλοι είναι ίσης ακτίνας (θέτοντας $R1=250$), προκύπτουν οι εξής ενδιαφέρουσες δημιουργίες, ανάλογα της τιμής της φάσης D (π.χ. $D=15$ ή $D=10$).



ΚΕΦΑΛΑΙΟ 9, Ακτινωτό σχήμα (5)

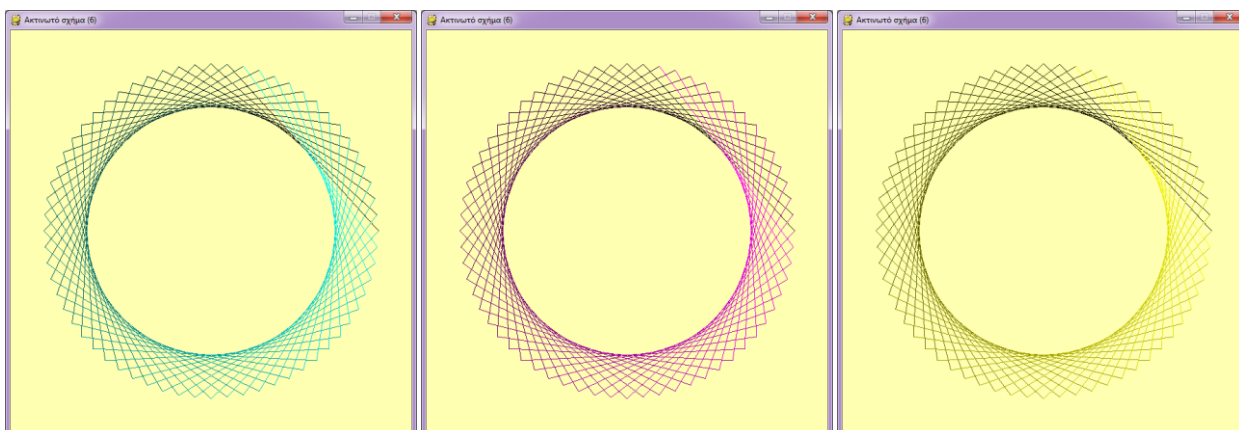
Αν θέλουμε οι N γραμμές να μην είναι όλες ίδιου χρώματος τότε πρέπει με κάποιο τρόπο να αλλάζουμε σε κάθε επανάληψη το όρισμα που καθορίζει το χρωματισμό της ακτίνας που σχεδιάζεται. Μια λύση είναι να χρησιμοποιήσουμε την έκφραση $i*256/N$ ως συνιστώσα ενός από τους τρεις βασικούς χρωματισμούς κόκκινο, πράσινο ή μπλε. Γράφουμε τον κώδικα που ακολουθεί και τον αποθηκεύουμε π.χ. ως 7Kallitexnies09.py.



Καταρχήν παρατηρεί κανείς ότι με την εντολή `epif.fill((255,255,177))` η επιφάνεια σχεδίασης `epif` απέκτησε έναν ανοιχτόχρωμο μπεζ χρωματισμό, χρήσιμο στο να διακρίνονται οι σκούρες γραμμές επάνω της. Στη συνέχεια το μόνο που άλλαξε σε σχέση με το προηγούμενο πρόγραμμα είναι το N το οποίο πήρε τιμή 64 και ο χρωματισμός. Η έκφραση $(i*256/N, 0, 0)$ δίνει στην κόκκινη συνιστώσα πρωταγωνιστικό ρόλο. Καθώς το i θα παίρνει διαδοχικά τις τιμές 0, 1, 2, ..., 63, η κόκκινη συνιστώσα θα γίνεται 0, 4, 8, ..., 252 αντίστοιχα. Δηλαδή θα σχεδιάζονται αρχικά σκούρες γραμμές, οι οποίες σιγά σιγά θα γίνονται πιο λαμπρές κόκκινες.

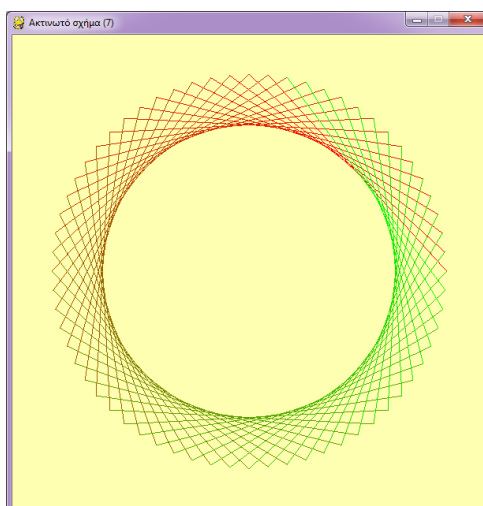
ΚΕΦΑΛΑΙΟ 10, Ακτινωτό σχήμα (6)

Δοκιμάζοντας την έκφραση $(0, i*256/N, i*256/N)$ θα δούμε σκούρες γραμμές σιγά σιγά να πηγαίνουν προς κυανούς (γαλάζιους) χρωματισμούς, δοκιμάζοντας την έκφραση $(i*256/N, 0, i*256/N)$ θα δούμε σκούρες γραμμές σιγά σιγά να πηγαίνουν προς ματζέντα χρωματισμούς και δοκιμάζοντας την έκφραση $(i*256/N, i*256/N, 0)$ θα δούμε σκούρες γραμμές σιγά σιγά να κιτρινίζουν. Το κυανό χρώμα καθορίζεται με τιμές $(0, 255, 255)$, το ματζέντα καθορίζεται με τιμές $(255, 0, 255)$ και το κίτρινο καθορίζεται με τιμές $(255, 255, 0)$ και προς αυτές τις τιμές τείνουν οι παραστάσεις των χρωματισμών καθώς αυξάνεται η τιμή του i σε κάθε επανάληψη. Αποθηκεύουμε π.χ. ως 7Kallitexnies10.py.



ΚΕΦΑΛΑΙΟ 11, Ακτινωτό σχήμα (7)

Δοκιμάζοντας την έκφραση $(255-i*256/N, i*256/N, 0)$ ως χρώμα γραμμής, παρατηρούμε ότι ο κόκκινος χρωματισμός σιγά σιγά θα φθίνει, ενώ ο πράσινος χρωματισμός σιγά σιγά θα αυξάνει. Έτσι, καθώς ουσιαστικά εκκινούμε από τον κόκκινο χρωματισμό $(255, 0, 0)$ και τερματίζουμε στο πράσινο χρωματισμό $(0, 255, 0)$, το αποτέλεσμα θα είναι το παρακάτω:

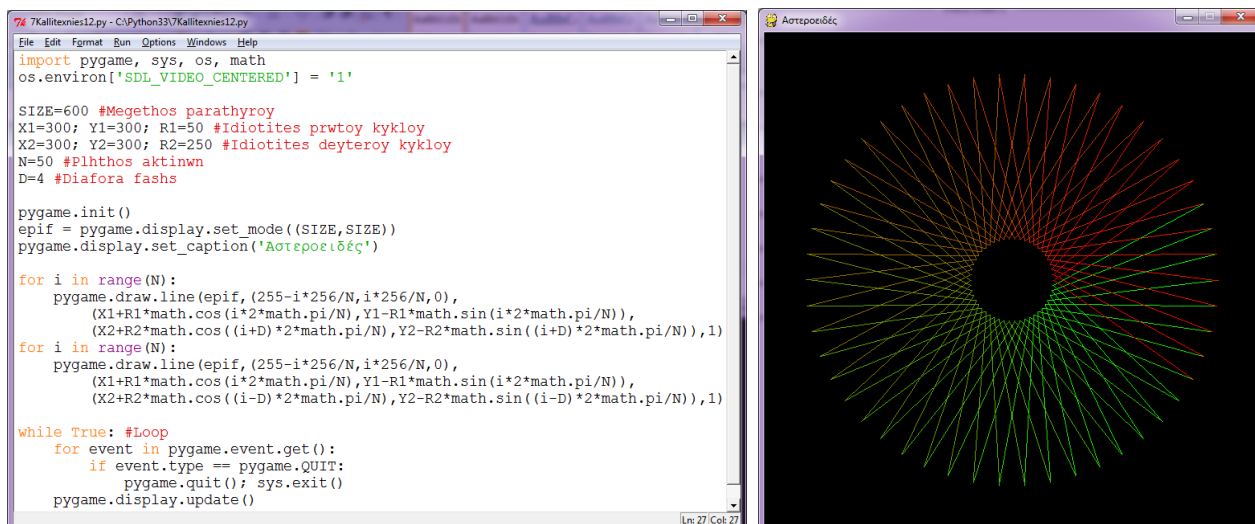


Αποθηκεύουμε π.χ. ως 7Kallitexnies11.py.

ΚΕΦΑΛΑΙΟ 12, Αστεροειδές

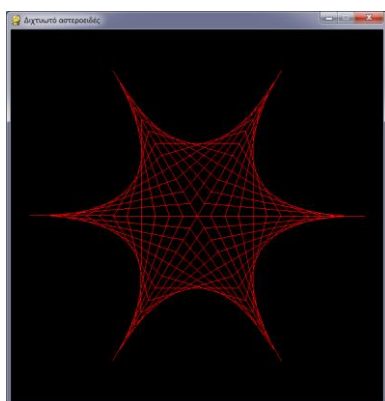
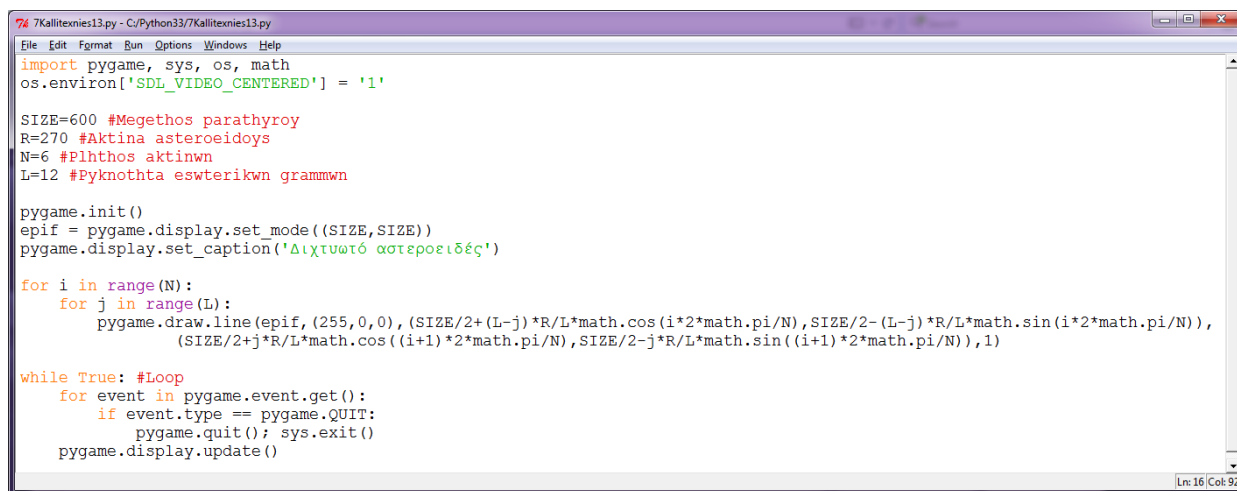
Μπορεί κανείς να σκεφτεί διάφορες δημιουργικές προεκτάσεις των παραπάνω προαναφερθέντων. Π.χ. διατηρώντας το προηγούμενο τέχνασμα της αλλαγής χρωματισμών σε συνδυασμό με διπλή εφαρμογή της εντολής `for` του κώδικα 7Kallitexnies08.py με αλλαγμένες τις παραμέτρους, προκύπτουν ενδιαφέρουσες δημιουργίες.

Πληκτρολογούμε τον παρακάτω κώδικα και αποθηκεύουμε π.χ. ως 7Kallitexnies12.py



ΚΕΦΑΛΑΙΟ 13, Διχτυωτό αστεροειδές

Γράφουμε τον παρακάτω κώδικα 7Kallitexnies13.py και τον εκτελούμε.



Φαίνεται αρκετά περίπλοκος ο κώδικας, όμως θα γίνει προσπάθεια να αναλυθεί και να εξηγηθεί. Η έκφραση R/L που εμφανίζεται συχνά, είναι ένα στοιχειώδες μήκος τμήματος στο δίκτυο. Για κάθε i , θα σχεδιαστούν L γραμμές οι οποίες θα εκκινούν όλες από σημεία κύκλου ακτίνας $(L-j) * R/L$ και θα τερματίζουν σε σημεία κύκλου ακτίνας $j * R/L$. Όταν το j είναι 0 στην πρώτη επανάληψη, θα σχεδιάζονται γραμμές από τις κορυφές των ακτίνων προς το κέντρο. Όταν το j γίνεται μέγιστο (δηλαδή $L-1$) στην τελευταία επανάληψη θα σχεδιάζονται γραμμές από κύκλο ακτίνας R/L προς τις κορυφές της διπλανής ακτίνας. Για j μεταξύ 0 και $L-1$ σχεδιάζονται οι ενδιάμεσες γραμμές που σχηματίζουν το πύκνωμα. Όταν $i=0$ παρατηρούμε ότι θα σχηματιστούν οι γωνίες 0 και $2\pi/N$, άρα αν $N=6$ όλες οι γραμμές που θα σχεδιαστούν στην εντολή for του j , θα εκκινούν και θα τερματίζουν στις γραμμές-ακτίνες που θα σχημάτιζαν ο ωροδείκτης στις 3:00 και στις 1:00. Όταν $i=1$ παρατηρούμε ότι θα σχηματιστούν οι γωνίες $2\pi/N$ και $2*2\pi/N$, άρα αν $N=6$ όλες οι γραμμές που θα σχεδιαστούν στην εντολή for του j , θα εκκινούν και θα τερματίζουν στις γραμμές-ακτίνες που θα

σχημάτιζαν ο ωροδείκτης στις 1:00 και στις 11:00. Αξίζει να δοκιμάσει κανείς να τρέξει το πρόγραμμα με αλλαγές στις τιμές του N και του L.

ΚΕΦΑΛΑΙΟ 14, Διχτυωτό αστεροειδές δίχρωμο

Γράφουμε τον παρακάτω κώδικα 7Kallitexnies14.py και τον εκτελούμε.

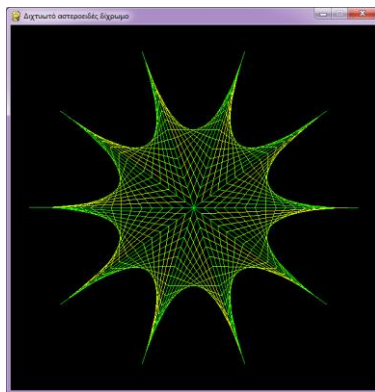
```
7Kallitexnies14.py - C:/Python33/7Kallitexnies14.py
File Edit Format Run Options Windows Help
import pygame, sys, os; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'

SIZE=600 #Megethos parathyroy
R=270 #Aktina asteroeidoy
N=10 #Plhthos aktinwn
L=16 #Pyknothta eswterikwn grammwn

pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Διχτυωτό αστεροειδές δίχρωμο')

for i in range(N):
    for j in range(L):
        pygame.draw.line(epif, (j*16,255,0), (SIZE/2+(L-j)*R/L*cos(i*2*pi/N), SIZE/2-(L-j)*R/L*sin(i*2*pi/N)),
            (SIZE/2+j*R/L*cos((i+1)*2*pi/N), SIZE/2-j*R/L*sin((i+1)*2*pi/N)),1)

while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.display.update()
```

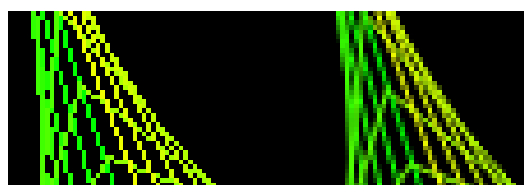


Καταρχήν παρατηρούμε ότι οι ακτίνες είναι δέκα καθώς τέθηκε $N=10$. Παρατηρούμε επίσης μια διχρωμία. Αυτή επετεύχθη με τον χρωματισμό $(j*16,255,0)$ κάτι που σημαίνει ότι καθώς το j παίρνει διαδοχικά τιμές από 0 έως 15, η τιμή του κόκκινου θα παίζει μεταξύ 0 και 240. Άρα, σε συνδυασμό με τιμή πράσινου συνεχώς στο μέγιστο 255, θα σχεδιάζονται γραμμές από $(0,255,0)$ έως $(240,255,0)$ δηλαδή γραμμές μεταξύ πράσινου και κίτρινου. Και αυτό θα γίνει N φορές, δηλαδή δέκα φορές.

Παρατηρούμε μια αλλαγή στην πρώτη γραμμή του κώδικα. Αντί να φορτώνεται ολόκληρο το πακέτο εντολών `math`, με την εντολή `from math import pi, sin, cos` φορτώνονται μόνο οι τρεις απαραίτητες εντολές κερδίζοντας έτσι σε πόρους μνήμης. Επίσης όμως, παρακάτω στην παρακάτω εντολή σχεδίασης, δεν χρειάζεται τώρα να χρησιμοποιούμε το πρόθεμα `math`, κερδίζοντας κάποιο χώρο σύνταξης κώδικα και διατηρώντας τον ευανάγνωστο.

ΚΕΦΑΛΑΙΟ 15, Η εντολή `pygame.draw.aaline`

Στο προηγούμενο πρόγραμμα πραγματοποιούμε μόνο μια απλή αλλαγή: Αντικαθιστούμε την εντολή `pygame.draw.line` με `pygame.draw.aaline`. Τα γράμματα `aa` υποδηλώνουν τη φράση **anti-aliasing**. Το αποτέλεσμα ίσως φαίνεται μόνο στη μεγέθυνση, πάντως οι γραμμές σχεδιάζονται πράγματι πιο προσαρμοσμένες στο εκάστοτε φόντο ειδικά όταν πρόκειται για διαγώνιες γραμμές όπου τα πιξελ σχηματίζουν ενοχλητικές μικροσκοπικές γωνίες. Αποθηκεύουμε π.χ. ως 7Kallitexnies15.py.



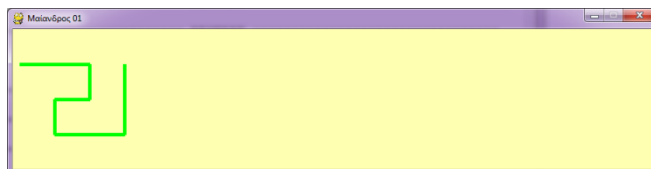
ΚΕΦΑΛΑΙΟ 16, Μαϊάνδρος 01

Γράφουμε τον παρακάτω κώδικα 7Kallitexnies16.py και τον εκτελούμε.

```
7Kallitexnies16.py - C:\Python33\7Kallitexnies16.py
File Edit Format Run Options Windows Help
import pygame, sys, os
os.environ['SDL_VIDEO_CENTERED'] = '1'

pygame.init()
epif = pygame.display.set_mode((920,200))
pygame.display.set_caption('Μαϊάνδρος 01')
epif.fill((255,255,177)) #Xrwmatismos anoixtoxrwmos mpez

x=10; y=50 #Arxh toy maiandroy
pygame.draw.line(epif, (0,255,0), (x,y), (x+100,y), 5)
pygame.draw.line(epif, (0,255,0), (x+100,y), (x+100,y+50), 5)
pygame.draw.line(epif, (0,255,0), (x+100,y+50), (x+50,y+50), 5)
pygame.draw.line(epif, (0,255,0), (x+50,y+50), (x+50,y+100), 5)
pygame.draw.line(epif, (0,255,0), (x+50,y+100), (x+150,y+100), 5)
pygame.draw.line(epif, (0,255,0), (x+150,y+100), (x+150,y), 5)
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.display.update()
```



Δεν χρειάζονται πολλές επεξηγήσεις. Σχεδιάζονται απλά έξι γραμμές, σχηματίζοντας ένα τμήμα μαιάνδρου. Ας προσπαθήσουμε να επεκτείνουμε το πρόγραμμα.

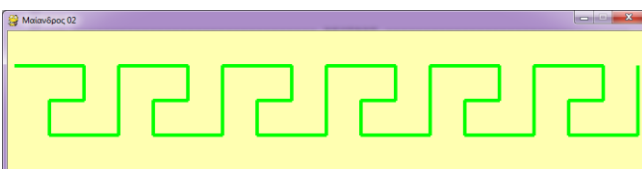
ΚΕΦΑΛΑΙΟ 17, Μαϊάνδρος 02

Τροποποιούμε τον παραπάνω κώδικα, αποθηκεύουμε ως 7Kallitexnies17.py και εκτελούμε.

```
7Kallitexnies17.py - C:\Python33\7Kallitexnies17.py
File Edit Format Run Options Windows Help
import pygame, sys, os
os.environ['SDL_VIDEO_CENTERED'] = '1'

pygame.init()
epif = pygame.display.set_mode((920,200))
pygame.display.set_caption('Μαϊάνδρος 02')
epif.fill((255,255,177)) #Xrwmatismos anoixtoxrwmos mpez

x=10; y=50 #Arxh toy maiandroy
for i in range(6):
    pygame.draw.line(epif, (0,255,0), (x,y), (x+100,y), 5)
    pygame.draw.line(epif, (0,255,0), (x+100,y), (x+100,y+50), 5)
    pygame.draw.line(epif, (0,255,0), (x+100,y+50), (x+50,y+50), 5)
    pygame.draw.line(epif, (0,255,0), (x+50,y+50), (x+50,y+100), 5)
    pygame.draw.line(epif, (0,255,0), (x+50,y+100), (x+150,y+100), 5)
    pygame.draw.line(epif, (0,255,0), (x+150,y+100), (x+150,y), 5)
    x=x+150
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.display.update()
```



Παρατηρούμε ότι οι εντολές που σχεδίαζαν προηγουμένως ένα τμήμα μαιάνδρου, τώρα έχουν τοποθετηθεί μέσα σε ένα εξαπλό επαναληπτικό for και έχει προστεθεί η εντολή **x=x+150**, ώστε κάθε επόμενο τμήμα μαιάνδρου να σχηματίζεται 150 πίξελ πιο δεξιά.

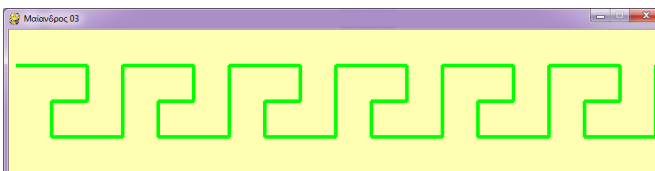
ΚΕΦΑΛΑΙΟ 18, Μαϊάνδρος 03

Τροποποιούμε τον παραπάνω κώδικα, αποθηκεύουμε ως 7Kallitexnies18.py και εκτελούμε.

```
7Kallitexnies18.py - C:\Python33\7Kallitexnies18.py
File Edit Format Run Options Windows Help
import pygame, sys, os
os.environ['SDL_VIDEO_CENTERED'] = '1'

pygame.init()
epif = pygame.display.set_mode((920,200))
pygame.display.set_caption('Μαϊάνδρος 03')
epif.fill((255,255,177)) #Xrwmatismos anoixtoxrwmos mpez

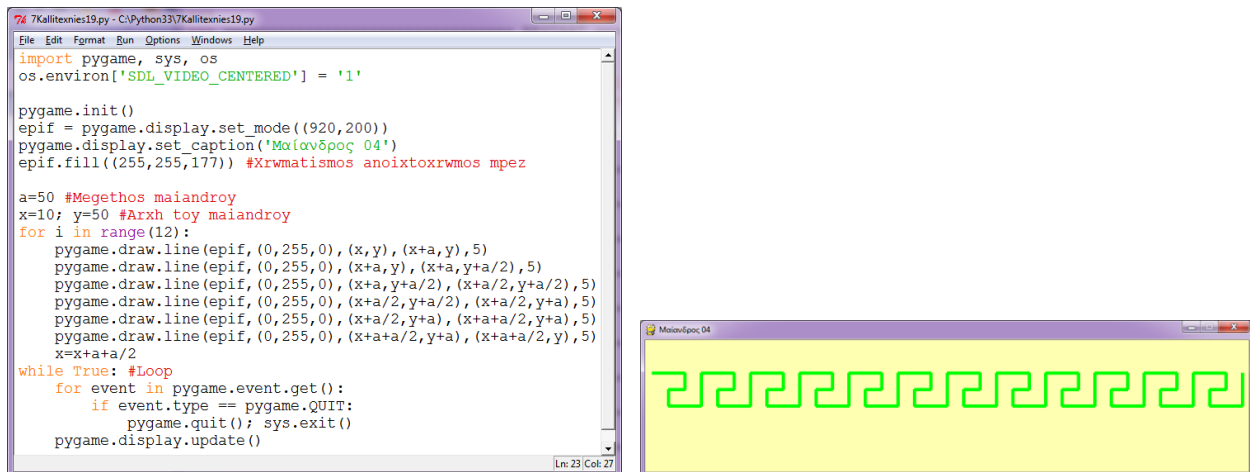
a=100 #Megethos maiandroy
x=10; y=50 #Arxh toy maiandroy
for i in range(6):
    pygame.draw.line(epif, (0,255,0), (x,y), (x+a,y), 5)
    pygame.draw.line(epif, (0,255,0), (x+a,y), (x+a,y+a/2), 5)
    pygame.draw.line(epif, (0,255,0), (x+a,y+a/2), (x+a/2,y+a/2), 5)
    pygame.draw.line(epif, (0,255,0), (x+a/2,y+a/2), (x+a/2,y+a), 5)
    pygame.draw.line(epif, (0,255,0), (x+a/2,y+a), (x+a+a/2,y+a), 5)
    pygame.draw.line(epif, (0,255,0), (x+a+a/2,y+a), (x+a+a/2,y), 5)
    x=x+a+a/2
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.display.update()
```



Δεν υπάρχει διαφορά στην εκτέλεση σε σχέση με τον προηγούμενο κώδικα. Όμως, προσέχοντας τον κώδικα, παρατηρούμε ότι έχει επιτευχθεί η λεγόμενη γενίκευση. Το παρόν πρόγραμμα είναι πιο γενικό, δηλαδή μπορεί εύκολα να παράγει μικρότερου ή μεγαλύτερου μεγέθους μαιάνδρους ανάλογα της τιμής του a . Προστέθηκε η εντολή $a=100$ και στις εντολές `pygame.draw.line` αντικαταστάθηκε κάθε εμφάνιση του 100 με a , κάθε εμφάνιση του 50 με $a/2$ και κάθε εμφάνιση του 150 με $a+a/2$.

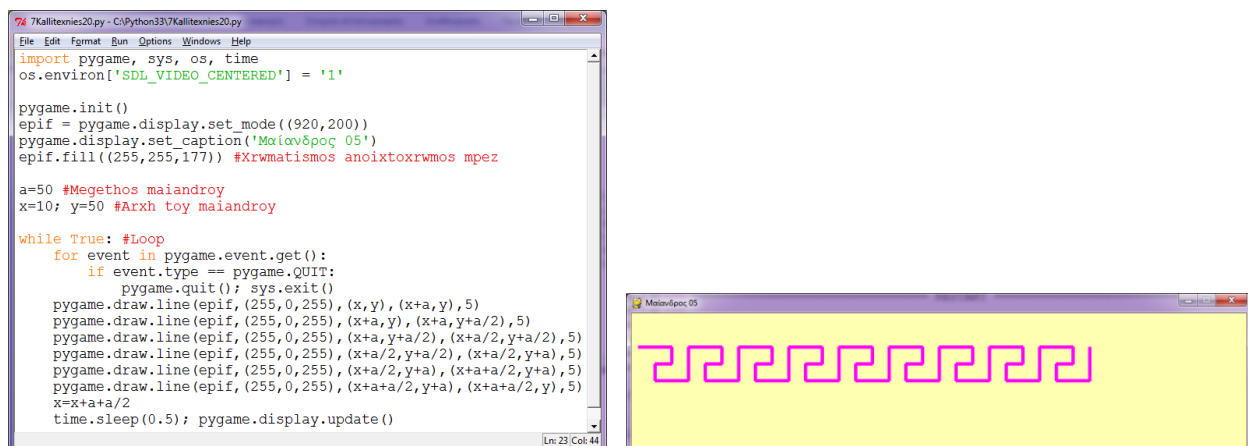
ΚΕΦΑΛΑΙΟ 19, Μαιάνδρος 04

Τροποποιούμε ελαφρά τον παραπάνω κώδικα θέτοντας $a=50$ και αλλάζοντας τις επαναλήψεις στο `for` από 6 σε 12. Αποθηκεύουμε ως 7Kallitexnies19.py και εκτελούμε.



ΚΕΦΑΛΑΙΟ 20, Μαιάνδρος 05

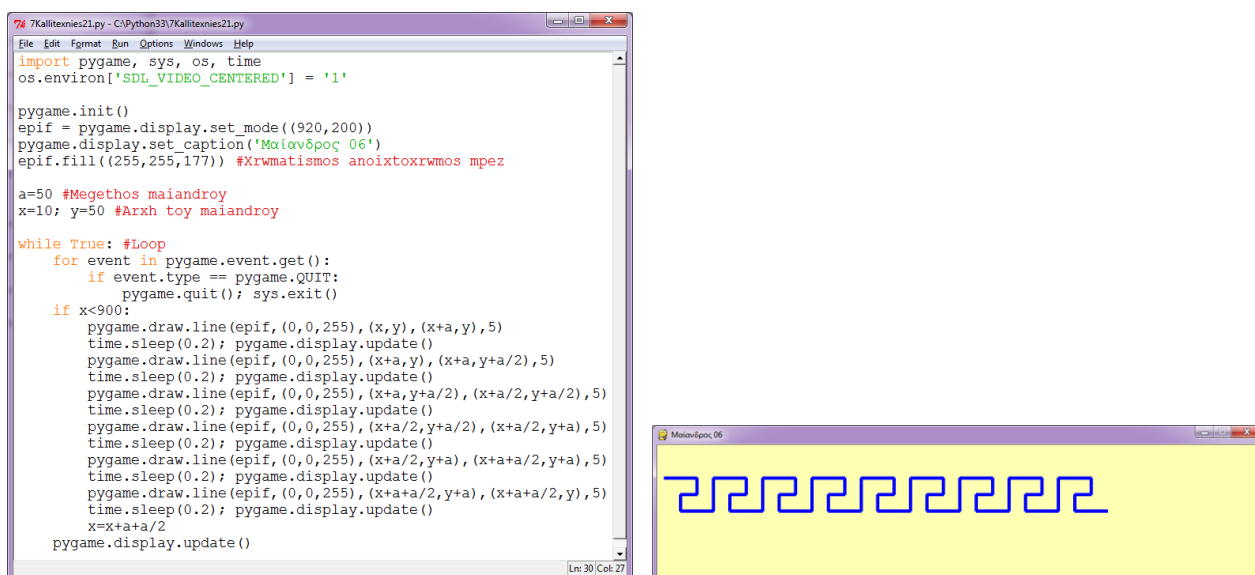
Σε αυτό το κεφάλαιο, στόχος μας είναι να προσθέσουμε τον παράγοντα κίνηση στο χρόνο. Τροποποιούμε τον παραπάνω κώδικα προσθέτοντας το module `time` στην πρώτη γραμμή κώδικα, θέτοντας χρώμα ματζέντα (255,0,255), μεταφέροντας τις εντολές σχεδίασης εντός του Loop και προσθέτοντας την εντολή `time.sleep(0.5)` πριν από την τελευταία εντολή `pygame.display.update()`. Αποθηκεύουμε ως 7Kallitexnies20.py και εκτελούμε.



Παρατηρούμε ότι πλέον δεν σχεδιάζεται αστραπιαία ο μαιάνδρος. Μεταξύ της σχεδίασης κάθε τμήματος μεσολαβεί μισό δευτερόλεπτο. Ο χρόνος ορίστηκε στην εντολή `time.sleep` όπου το όρισμα αναφέρεται σε δευτερόλεπτα.

ΚΕΦΑΛΑΙΟ 21, Μαϊάνδρος 06

Στο προηγούμενο πρόγραμμα σχεδιάζονται αέναα τμήματα μαιάνδρου ακόμα και όταν αυτά πλέον δεν φαίνονται. Γι' αυτό το λόγο, τροποποιούμε τον παραπάνω κώδικα μεταφέροντας τις εντολές σχεδίασης εντός της εντολής `if x<900:`, κάτι που σημαίνει ότι αυτές πρέπει να γραφούν παρακάτω σε εσοχή σε σχέση με την `if`. Έτσι, μόνο αν η οριζόντια συντεταγμένη x του τμήματος μαιάνδρου βρίσκεται εντός καμβά ($x<900$), το τμήμα αυτό σχεδιάζεται. Προσθέτουμε επίσης την εντολή `time.sleep(0.2)` ακολουθούμενη από την εντολή `pygame.display.update()` έπειτα από κάθε εντολή σχεδίασης γραμμής. Έτσι πετυχαίνουμε τις μικρές χρονικές διακοπές σε κάθε γραμμή και όχι σε κάθε τμήμα. Αλλάζουμε επίσης αν θέλουμε το χρώμα σε μπλε και το χρόνο διακοπών σε 0.2 δευτερόλεπτα. Αποθηκεύουμε ως 7Kallitexnies21.py και εκτελούμε.

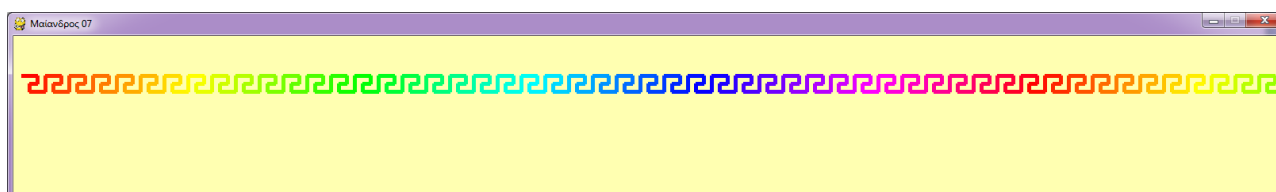


Για λόγους συμμαζέματος του κώδικα σε λίγες γραμμές, μπορούμε να γράψουμε τις εντολές σχεδίασης μαζί με τις εντολές `time.sleep(0.2)` και `pygame.display.update()`, αρκεί να τις διαχωρίζουμε με το ερωτηματικό.

```
if x<900:
    pygame.draw.line(epif, (0,0,255), (x,y), (x+a,y), 5); time.sleep(0.2); pygame.display.update()
    pygame.draw.line(epif, (0,0,255), (x+a,y), (x+a,y+a/2), 5); time.sleep(0.2); pygame.display.update()
    pygame.draw.line(epif, (0,0,255), (x+a,y+a/2), (x+a/2,y+a/2), 5); time.sleep(0.2); pygame.display.update()
    pygame.draw.line(epif, (0,0,255), (x+a/2,y+a/2), (x+a/2,y+a), 5); time.sleep(0.2); pygame.display.update()
    pygame.draw.line(epif, (0,0,255), (x+a/2,y+a), (x+a+a/2,y+a), 5); time.sleep(0.2); pygame.display.update()
    pygame.draw.line(epif, (0,0,255), (x+a+a/2,y+a), (x+a+a/2,y), 5); time.sleep(0.2); pygame.display.update()
```

ΚΕΦΑΛΑΙΟ 22, Μαϊάνδρος 07

Σκοπός σε αυτό το κεφάλαιο είναι να δούμε την τεχνική με την οποία πετυχαίνουμε συνεχείς αλλαγές χρωματισμών στις γραμμές του μαιάνδρου.



Στα κεφάλαια 10 με 14 έχουμε ήδη εξετάσει το βασικό μυστικό του ζητήματος.

Αυτό δεν είναι άλλο από τη συνεχή μεταβολή του χρωματισμού, χρησιμοποιώντας μεταβαλλόμενα ορίσματα χρωματισμών στις εντολές `pygame.draw.line`. Ο παρακάτω πίνακας θα μας βοηθήσει περισσότερο στην κατανόηση.

		R	G	B
R	κόκκινο	255	0	0
Y	κίτρινο	255	255	0
G	πράσινο	0	255	0
C	γαλάζιο	0	255	255
B	μπλε	0	0	255
M	ματζέντα	255	0	255
R	κόκκινο	255	0	0

Παρατηρεί κανείς ότι κάθε χρώμα διαφέρει από το επόμενο του κατά 255 σε ένα από τα τρία μέρη. Για να μεταβούμε από το κόκκινο στο κίτρινο πρέπει σιγά σιγά να αυξήσουμε την πράσινη συνιστώσα. Για να μεταβούμε από το κίτρινο στο πράσινο πρέπει σιγά σιγά να μειώσουμε την κόκκινη συνιστώσα. Για να μεταβούμε από το πράσινο στο κυανό πρέπει σιγά σιγά να αυξήσουμε την μπλε συνιστώσα κτλ. Ας γράψουμε τον παρακάτω κώδικα `7Kallitexnies22.py`, ο οποίος πετυχαίνει το ζητούμενο:

```

74 7Kallitexnies22.py - C:/Python33/7Kallitexnies22.py
File Edit Format Run Options Windows Help
import pygame, sys, os, time
os.environ['SDL_VIDEO_CENTERED'] = '1'

def xrwma(a):
    '''Dexetai enan akeraio arithmo a kai dhmioyrgei thn antistoixh apoxrwsh.
    Geitonikoι arithmoι exoyn syggenikes apoxrwseis me poreia apoxrwsewn
    kokkino, kitrino, prasino, galazio, mple, matzenta, kokkino ktl. .'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akeraioMeros = a // 43 #To 43 xrhsimopoietai dioti moirazei sto peripoy
    ypoloipoDiareshs = a % 43 #thn perioxh 0 ews 255 se 6 isa merh.
    if akeraioMeros == 0: xrwmatismos = (255, ypoloipoDiareshs*255/43, 0)
    elif akeraioMeros == 1: xrwmatismos = (255-ypoloipoDiareshs*255/43, 255, 0)
    elif akeraioMeros == 2: xrwmatismos = (0, 255, ypoloipoDiareshs*255/43)
    elif akeraioMeros == 3: xrwmatismos = (0, 255-ypoloipoDiareshs*255/43, 255)
    elif akeraioMeros == 4: xrwmatismos = (ypoloipoDiareshs*255/43, 0, 255)
    elif akeraioMeros == 5: xrwmatismos = (255, 0, 255-ypoloipoDiareshs*255/43)
    return xrwmatismos

pygame.init()
epif = pygame.display.set_mode((1600,200))
pygame.display.set_caption('Μα(ανδρος 07')
epif.fill((255,255,177)) #Xrwmatismos anoixtoxrwmos mpez
a=20 #Megethos malandroy
x=10; y=50 #Arxh toy malandroy
c=1 #Akeraios arithmos poy tha ayksanetai synexeia kai tha metatrepetai se xrwma
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    if x<1600:
        pygame.draw.line(epif,xrwma(c),(x,y),(x+a,y),5); time.sleep(0.1); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(x+a,y),(x+a,y+a/2),5); time.sleep(0.1); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(x+a,y+a/2),(x+a/2,y+a/2),5); time.sleep(0.1); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(x+a/2,y+a/2),(x+a/2,y+a),5); time.sleep(0.1); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(x+a/2,y+a),(x+a+a/2,y+a),5); time.sleep(0.1); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(x+a+a/2,y+a),(x+a+a/2,y),5); time.sleep(0.1); pygame.display.update(); c=c+1
        x=x+a+a/2
    pygame.display.update()

```

Σε σχέση με το προηγούμενο πρόγραμμα, έχει προστεθεί ένα ολόκληρο μπλοκ εντολών στην τρίτη γραμμή, κάτι που ονομάζεται υποπρόγραμμα καθώς εκκινεί με `def`. Το υποπρόγραμμα ονομάζεται `xrwma` και έχει ως αποστολή να μετατρέπει έναν ακέραιο σε έναν κατάλληλο χρωματισμό δίνοντας σε γειτονικούς ακέραιους, παραπλήσιες αποχρώσεις.

Επειδή χρησιμοποιούνται εδώ, δυο λόγια για τους τελεστές // και % της Python, που ονομάζονται και τελεστής ακέραιας διαίρεσης και τελεστής υπολοίπου αντίστοιχα (ονομάζονται επίσης `div` και `mod`). Εφαρμόζονται σε δύο αριθμούς. Όταν ο `//` εφαρμόζεται

σε δύο ακέραιους θετικούς, επιστρέφει το ακέραιο μέρος του πηλίκου τους. Όταν ο % εφαρμόζεται σε δύο ακέραιους θετικούς, επιστρέφει το ακέραιο υπόλοιπο της διαίρεσής τους. Καλό είναι να πειραματιστούμε στο κέλυφος της Pythοn, πληκτρολογώντας π.χ. τα εξής:

```
>>> 5 // 2
2
>>> 7 // 2
3
>>> 10 // 3
3
>>> 11 // 3
3
>>> 100 // 40
2
>>>
>>> 5 % 2
1
>>> 7 % 2
1
>>> 10 % 3
1
>>> 11 % 3
2
>>> 100 % 40
20
>>>
```

Για παράδειγμα στην τελευταία πράξη div της πρώτης στήλης υπολογίζεται πόσες ακέραιες φορές χωράει το 40 στο 100 ενώ στην τελευταία πράξη mod της δεύτερης στήλης επιστρέφεται το 20 καθώς αυτό περισεύει μετά την ακέραια διαίρεση των αριθμών 100 και 40.

Γιατί όμως είναι χρήσιμοι αυτοί οι τελεστές στο πρόγραμμά μας; Ουσιαστικά, διότι με τη χρήση τους μπορούν εύκολα να αντιστοιχισθούν συνεχόμενοι ακέραιοι σε συνεχόμενες αποχρώσεις σύμφωνα με το παραπάνω σύστημα. Κάθε φορά, στον ακέραιο αριθμό *a* εφαρμόζεται η πράξη div 43 και η πράξη mod 43. Για τους ακέραιους αριθμούς από το 1 έως το 42, το αποτέλεσμα div θα είναι πάντα 0 (καθώς δεν χωράει το 43) ενώ το αποτέλεσμα mod θα είναι οι αριθμοί 1 έως 42 (καθώς ο αριθμός περισεύει ολόκληρος). Για τους ακέραιους αριθμούς από το 43 έως το 85, το αποτέλεσμα div θα είναι πάντα 1 (καθώς το 43 χωράει μια φορά), ενώ το αποτέλεσμα mod θα είναι οι αριθμοί 1 έως 42 (δηλαδή το επιπλέον μέρος πέραν του 43). Για τους ακέραιους αριθμούς από το 86 έως το 128, το αποτέλεσμα div θα είναι πάντα 2 (καθώς το 43 χωράει δύο φορές) ενώ το αποτέλεσμα mod θα είναι οι αριθμοί 1 έως 42 (δηλαδή το επιπλέον μέρος πέραν του 86). Και ούτω καθεξής. Παρατηρούμε ότι όταν ο αριθμός βρίσκεται στην πρώτη περιοχή, προκαλούνται χρωματισμοί από το κόκκινο προς το κίτρινο, διότι η παράσταση `(255, υπολοιποDiaireshs*255/43, 0)` παίρνει περίπου τιμές από το (255,0,0) συγκλίνουσες προς το (255,255,0). Στη συνέχεια, όταν ο αριθμός βρίσκεται στη δεύτερη περιοχή, προκαλούνται χρωματισμοί από το κίτρινο προς το πράσινο, διότι η παράσταση `(255-υπολοιποDiaireshs*255/43, 255, 0)` παίρνει περίπου τιμές από το (255,255,0) συγκλίνουσες προς το (0,255,0). Είναι αξιοσημείωτο ότι όταν ο αριθμός βρίσκεται στην έκτη περιοχή (δηλαδή από το 216 και μετά, όπου το αποτέλεσμα div είναι 5), προκαλούνται χρωματισμοί από το ματζέντα προς το αρχικό κόκκινο, διότι η παράσταση `(255, 0, 255-υπολοιποDiaireshs*255/43)` παίρνει περίπου τιμές από το (255,0,255) συγκλίνουσες προς το (255,0,0). Έτσι επανέρχονται οι χρωματισμοί στο κόκκινο. Οι αριθμοί μετά το 255, ουσιαστικά δημιουργούν χρωματισμούς ισοδύναμους με τους χρωματισμούς των προηγούμενων 255 αριθμών και για αυτό φροντίζει η εντολή `a = a % 255`.

Πρέπει να τονισθεί ότι το υποπρόγραμμα `xrwna` καλείται σε κάθε εντολή σχεδίασης, καθώς βλέπουμε στο σημείο του χρωματισμού την παράσταση `xrwna(c)`. Όπως βλέπουμε, η μεταβλητή *c* εκκινεί από το 1 και έπειτα από κάθε εντολή σχεδίασης γραμμής, αυξάνεται η τιμή της κατά 1. Κατά την κλήση του υποπρογράμματος, μεταβιβάζεται η τιμή της *c* ως τιμή της μεταβλητής *a* του υποπρογράμματος. Εκεί γίνεται ο υπολογισμός του αντίστοιχου χρωματισμού, ο οποίος επιστρέφεται από το υποπρόγραμμα στο κυρίως πρόγραμμα στο σημείο της αντίστοιχης εντολής σχεδίασης γραμμής.

ΚΕΦΑΛΑΙΟ 23, Μαϊάνδρος 08

Σκοπός σε αυτό το κεφάλαιο είναι να δείξουμε έναν τρόπο για σχεδίαση κυκλικού μαϊάνδρου. Γράφουμε τον παρακάτω κώδικα 7Kallitexnies23.py και τον τρέχουμε.

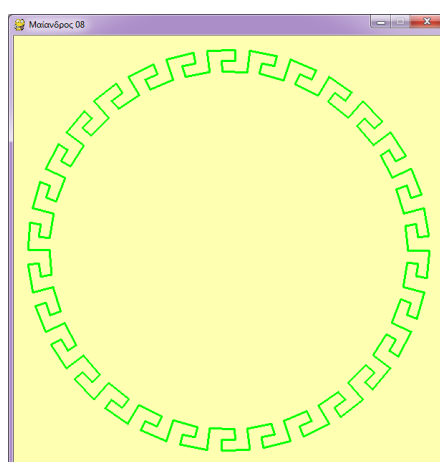
```
7Kallitexnies23.py - C:\Python33\7Kallitexnies23.py
File Edit Format Run Options Windows Help
import pygame, sys, os; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'

SIZE=600 #Megethos parathyroy
R=250 #Aktina kykloy
P=30 #Pachos maiandroy
N=30 #Plhthos tmhmatwn maiandroy se kyklo
N3=N*3 #Plhthos gwniwn se kyklo

pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Μαϊάνδρος 08')
epif.fill((255,255,177)) #Xrwmatismos anoixtoxrwmos mpez

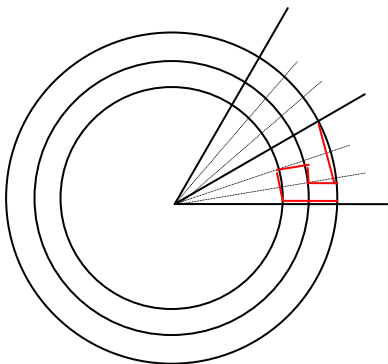
for i in range(N):
    pygame.draw.line(epif, (0,255,0),
        (SIZE/2+(R+P)*cos(3*i*2*pi/N3), SIZE/2-(R+P)*sin(3*i*2*pi/N3)),
        (SIZE/2+R*cos(3*i*2*pi/N3), SIZE/2-R*sin(3*i*2*pi/N3)), 3)
    pygame.draw.line(epif, (0,255,0),
        (SIZE/2+R*cos(3*i*2*pi/N3), SIZE/2-R*sin(3*i*2*pi/N3)),
        (SIZE/2+R*cos((3*i+2)*2*pi/N3), SIZE/2-R*sin((3*i+2)*2*pi/N3)), 3)
    pygame.draw.line(epif, (0,255,0),
        (SIZE/2+R*cos((3*i+2)*2*pi/N3), SIZE/2-R*sin((3*i+2)*2*pi/N3)),
        (SIZE/2+(R+P/2)*cos((3*i+2)*2*pi/N3), SIZE/2-(R+P/2)*sin((3*i+2)*2*pi/N3)), 3)
    pygame.draw.line(epif, (0,255,0),
        (SIZE/2+(R+P/2)*cos((3*i+2)*2*pi/N3), SIZE/2-(R+P/2)*sin((3*i+2)*2*pi/N3)),
        (SIZE/2+(R+P/2)*cos((3*i+1)*2*pi/N3), SIZE/2-(R+P/2)*sin((3*i+1)*2*pi/N3)), 3)
    pygame.draw.line(epif, (0,255,0),
        (SIZE/2+(R+P/2)*cos((3*i+1)*2*pi/N3), SIZE/2-(R+P/2)*sin((3*i+1)*2*pi/N3)),
        (SIZE/2+(R+P)*cos((3*i+1)*2*pi/N3), SIZE/2-(R+P)*sin((3*i+1)*2*pi/N3)), 3)
    pygame.draw.line(epif, (0,255,0),
        (SIZE/2+(R+P)*cos((3*i+1)*2*pi/N3), SIZE/2-(R+P)*sin((3*i+1)*2*pi/N3)),
        (SIZE/2+(R+P)*cos((3*i+3)*2*pi/N3), SIZE/2-(R+P)*sin((3*i+3)*2*pi/N3)), 3)

while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.display.update()
```



Ουσιαστικά χρησιμοποιήθηκε η τεχνική που έχει εφαρμοστεί και στο ακτινωτό σχήμα του κεφαλαίου 7. Όμως υπάρχουν διαφορές. Εδώ χρησιμοποιήθηκε ως κέντρο του κύκλου το κέντρο του παραθύρου, του οποίου οι δύο συντεταγμένες έχουν και οι δύο την τιμή $SIZE/2$. Αν και το πλήθος των μαϊάνδρων στον κύκλο είναι N , ο κύκλος έσπασε σε $N*3$ τμήματα και γι' αυτό το λόγο βλέπουμε συνεχώς την παράσταση $2*pi/N3$ ($N3=N*3$). Αυτή η παράσταση

πολλαπλασιάζεται είτε με $3*i$, είτε με $3*i+1$, είτε με $3*i+2$ ανάλογα σε ποια γωνία βρίσκεται το σημείο που πρέπει να αναφερθεί. Το παρακάτω σχήμα μας διαφωτίζει.

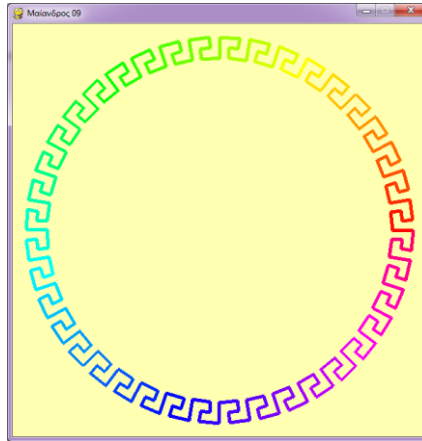


Επίσης, ανάλογα σε ποιον από τους τρεις κύκλους βρίσκεται το σημείο που πρέπει να αναφερθεί, το συνημίτονο και το ημίτονο θα πολλαπλασιαστεί είτε με R (ο μικρότερος κύκλος), είτε με $R+P/2$ (ο μεσαίος κύκλος), είτε με $R+P$ (ο μεγαλύτερος κύκλος).

ΚΕΦΑΛΑΙΟ 24, Μαϊάνδρος 09

Πώς μπορούμε να έχουμε σταδιακή σχεδίαση κυκλικού μαιάνδρου και μάλιστα με αλλαγή στους χρωματισμούς; Γράφουμε τον παρακάτω κώδικα 7Kallitexnies24.py και τον τρέχουμε.

```
import pygame, sys, os, time; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'
def xrwma(a):
    '''Dexetai enan akeraio arithmo a kai dhmiourgei thn antistoixh apoxrwsh. Geitonikoi arithmoi exoyn
    syggenikes apoxrweiseis me poreia apoxrwsewn kokkino, kitrino, prasino, galazio, mple, matzenta, kokkino ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akeraioMeros = a // 43 #To 43 xrhsimopoeitai dioti moirazei sto peripoy
    ypoloipoDiareshs = a % 43 #thn perioxh 0 ews 255 se 6 isa merh.
    if akeraioMeros == 0: xrwmatismos = (255, ypoloipoDiareshs*255/43, 0)
    elif akeraioMeros == 1: xrwmatismos = (255-ypoloipoDiareshs*255/43, 255, 0)
    elif akeraioMeros == 2: xrwmatismos = (0, 255, ypoloipoDiareshs*255/43)
    elif akeraioMeros == 3: xrwmatismos = (0, 255-ypoloipoDiareshs*255/43, 255)
    elif akeraioMeros == 4: xrwmatismos = (ypoloipoDiareshs*255/43, 0, 255)
    elif akeraioMeros == 5: xrwmatismos = (255, 0, 255-ypoloipoDiareshs*255/43)
    return xrwmatismos
SIZE=600 #Megethos parathyroy
R=250 #Aktina kykloy
P=30 #Pachos maiandroy
N=40 #Plhthos tmhmatwn maiandroy se kyklo
N3=N*3 #Plhthos gwniwn se kyklo
PG=5 #Pachos grammhs
DT=0.03 #Xronos diakophs se deyteerolepta
pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Μαϊάνδρος 09')
epif.fill((255,255,177)) #Xrwmatismos ανοιχτοxrwmos mpez
i=0 #Metrhths tmhmatwn maiandroy
c=1 #Akeraios arithmos poy tha ayksanetai synexeia kai tha metatrepetai se xrwma
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    if i<N:
        pygame.draw.line(epif,xrwma(c),(SIZE/2+(R+P)*cos(3*i*2*pi/N3),SIZE/2-(R+P)*sin(3*i*2*pi/N3)),
            (SIZE/2+R*cos(3*i*2*pi/N3),SIZE/2-R*sin(3*i*2*pi/N3)),PG)
        time.sleep(DT); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(SIZE/2+R*cos(3*i*2*pi/N3),SIZE/2-R*sin(3*i*2*pi/N3)),
            (SIZE/2+R*cos((3*i+2)*2*pi/N3),SIZE/2-R*sin((3*i+2)*2*pi/N3)),PG)
        time.sleep(DT); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(SIZE/2+R*cos((3*i+2)*2*pi/N3),SIZE/2-R*sin((3*i+2)*2*pi/N3)),
            (SIZE/2+(R+P/2)*cos((3*i+2)*2*pi/N3),SIZE/2-(R+P/2)*sin((3*i+2)*2*pi/N3)),PG)
        time.sleep(DT); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(SIZE/2+(R+P/2)*cos((3*i+2)*2*pi/N3),SIZE/2-(R+P/2)*sin((3*i+2)*2*pi/N3)),
            (SIZE/2+(R+P/2)*cos((3*i+1)*2*pi/N3),SIZE/2-(R+P/2)*sin((3*i+1)*2*pi/N3)),PG)
        time.sleep(DT); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(SIZE/2+(R+P/2)*cos((3*i+1)*2*pi/N3),SIZE/2-(R+P/2)*sin((3*i+1)*2*pi/N3)),
            (SIZE/2+(R+P)*cos((3*i+1)*2*pi/N3),SIZE/2-(R+P)*sin((3*i+1)*2*pi/N3)),PG)
        time.sleep(DT); pygame.display.update(); c=c+1
        pygame.draw.line(epif,xrwma(c),(SIZE/2+(R+P)*cos((3*i+1)*2*pi/N3),SIZE/2-(R+P)*sin((3*i+1)*2*pi/N3)),
            (SIZE/2+(R+P)*cos((3*i+3)*2*pi/N3),SIZE/2-(R+P)*sin((3*i+3)*2*pi/N3)),PG)
        time.sleep(DT); pygame.display.update(); c=c+1
        i=i+1
    pygame.display.update()
```

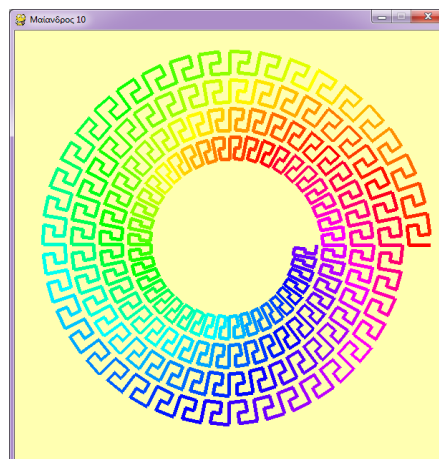



Ο κώδικας βασίζεται κυρίως στον κώδικα του 7Kallitexnies22.py, αλλά έγιναν κατάλληλες τροποποιήσεις, ώστε να χρησιμοποιηθούν οι τεχνικές του κώδικα 7Kallitexnies23.py. Παρατηρούμε ότι λόγω της κίνησης στο χρόνο απαιτείται να μεταφερθούν οι εντολές σχεδίασης γραμμών εντός του Loop. Ως χρόνος διακοπής χρησιμοποιείται η σταθερά DT με τιμή 0.03 δευτερόλεπτα, όμως μπορεί κανείς να δοκιμάσει και άλλες τιμές. Η σταθερά PG χρησιμοποιείται ως όρισμα πάχους γραμμής στις εντολές σχεδίασης με τιμή 5, αλλά και εδώ μπορεί κανείς να δοκιμάσει άλλες τιμές. Η επίτευξη της κυκλικότητας έχει εξηγηθεί στο κεφάλαιο 23, ενώ η αλλαγή χρωματισμού στο κεφάλαιο 22.

ΚΕΦΑΛΑΙΟ 25, Μαϊάνδρος 10

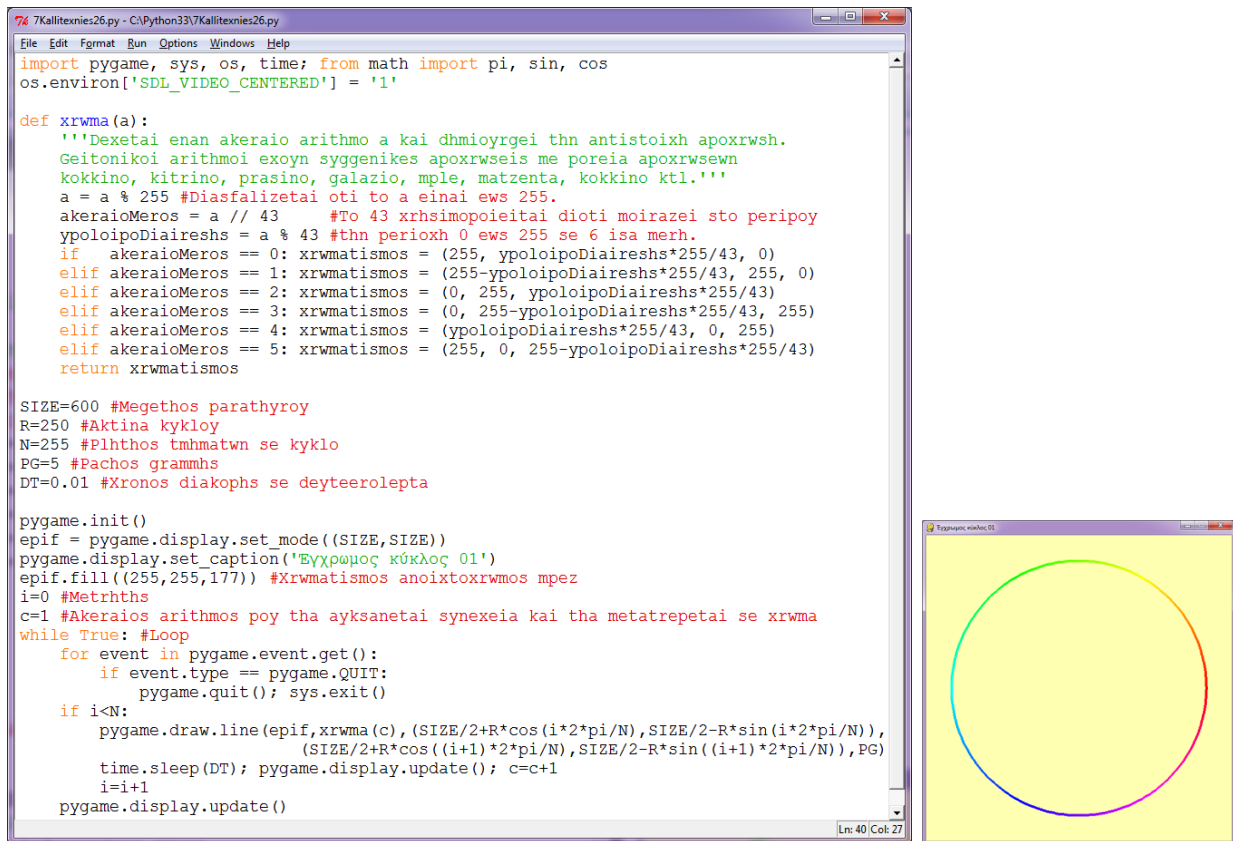
Τροποποιώντας πολύ ελαφρά τον προηγούμενο κώδικα μπορούμε να επιτύχουμε μαϊάνδρο σε σπирάλ. Αντί για `if i < N`: γράφουμε `if i < 4 * N`:. Αυτό γίνεται ώστε να σχεδιαστούν τμήματα μαϊάνδρου όχι μόνο σε μία αλλά σε τέσσερις κυκλικές περιστροφές. Επίσης προσθέτουμε κάτω από την εντολή `i=i+1`, την εντολή `R=R-1`. Αυτό γίνεται ώστε κάθε επόμενο τμήμα μαϊάνδρου, να σχεδιάζεται σε ελαφρά μικρότερο κύκλο.

Αποθηκεύουμε τον τροποποιημένο κώδικα ως 7Kallitexnies25.py και τον τρέχουμε.



ΚΕΦΑΛΑΙΟ 26, Έγχρωμος κύκλος 01

Ας απλοποιήσουμε τα πράγματα. Γράφουμε τον παρακάτω κώδικα 7Kallitexnies26.py, ο οποίος όταν εκτελείται, απλά διαγράφει μια έγχρωμη κυκλική γραμμή.



Αυτό που γίνεται είναι να σχεδιάζονται N γραμμές (στη περίπτωση μας 255), η μια πίσω απ' την άλλη, μεταβαλλόμενου χρωματισμού, δίνοντας την εντύπωση κανονικού κύκλου. Το αποτέλεσμα αυτό δεν μπορεί βέβαια να προέλθει με εντολή σχεδίασης κύκλου `pygame.draw.circle` διότι τότε θα είχαμε μονοχρωμία.

Αξιοσημείωτο ίσως είναι ότι για να εκκινούν και να τερματίζουν ομαλά οι χρωματισμοί στο κόκκινο, τα γραμμικά τμήματα που σχηματίζουν τον κύκλο είναι 255. Όσοι, δηλαδή, είναι και οι διαφορετικοί χρωματισμοί που επιστρέφει το υποπρόγραμμα `xrwma`.

ΚΕΦΑΛΑΙΟ 27, Έγχρωμος κύκλος 02

Μπορεί κανείς να παίξει με τους χρωματισμούς. Στο προηγούμενο παράδειγμα, οι χρωματισμοί διατρέχουν όλο το φάσμα όπως έχει επεξηγηθεί και στο κεφάλαιο 22. Όμως μπορούμε εύκολα να γράψουμε ένα άλλο υποπρόγραμμα χρωματισμών, το οποίο για παράδειγμα να παρακάμπτει τους μπλε χρωματισμούς, σύμφωνα με τον παρακάτω πίνακα.

		R	G	B
R	κόκκινο	255	0	0
Y	κίτρινο	255	255	0
M	ματζέντα	255	0	255
R	κόκκινο	255	0	0

Παρατηρεί κανείς ότι για να μεταβούμε από το κόκκινο στο κίτρινο πρέπει σιγά σιγά να αυξήσουμε την πράσινη συνιστώσα. Για να μεταβούμε από το κίτρινο στο ματζέντα πρέπει σιγά σιγά να μειώσουμε την πράσινη και παράλληλα να αυξήσουμε τη μπλε συνιστώσα. Για να μεταβούμε από το ματζέντα ξανά στο κόκκινο πρέπει σιγά σιγά να μειώσουμε την μπλε συνιστώσα κτλ. Ας παρατηρήσει κανείς τον παρακάτω πίνακα που παρουσιάζει επακριβώς αυτή τη διαδικασία με αριθμητικά δεδομένα.

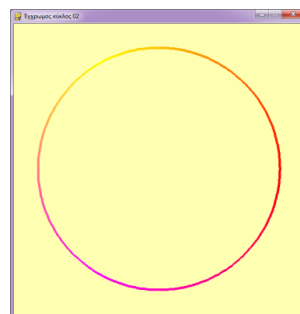
a	a div 85	a mod 85	R	G	B	a	a div 85	a mod 85	R	G	B	a	a div 85	a mod 85	R	G	B
1	0	1	255	3	0	86	1	1	255	252	3	171	2	1	255	0	252
2	0	2	255	6	0	87	1	2	255	249	6	172	2	2	255	0	249
3	0	3	255	9	0	88	1	3	255	246	9	173	2	3	255	0	246
4	0	4	255	12	0	89	1	4	255	243	12	174	2	4	255	0	243
5	0	5	255	15	0	90	1	5	255	240	15	175	2	5	255	0	240
6	0	6	255	18	0	91	1	6	255	237	18	176	2	6	255	0	237
7	0	7	255	21	0	92	1	7	255	234	21	177	2	7	255	0	234
8	0	8	255	24	0	93	1	8	255	231	24	178	2	8	255	0	231
9	0	9	255	27	0	94	1	9	255	228	27	179	2	9	255	0	228
10	0	10	255	30	0	95	1	10	255	225	30	180	2	10	255	0	225
11	0	11	255	33	0	96	1	11	255	222	33	181	2	11	255	0	222
12	0	12	255	36	0	97	1	12	255	219	36	182	2	12	255	0	219
13	0	13	255	39	0	98	1	13	255	216	39	183	2	13	255	0	216
14	0	14	255	42	0	99	1	14	255	213	42	184	2	14	255	0	213
15	0	15	255	45	0	100	1	15	255	210	45	185	2	15	255	0	210
16	0	16	255	48	0	101	1	16	255	207	48	186	2	16	255	0	207
17	0	17	255	51	0	102	1	17	255	204	51	187	2	17	255	0	204
18	0	18	255	54	0	103	1	18	255	201	54	188	2	18	255	0	201
19	0	19	255	57	0	104	1	19	255	198	57	189	2	19	255	0	198
20	0	20	255	60	0	105	1	20	255	195	60	190	2	20	255	0	195
21	0	21	255	63	0	106	1	21	255	192	63	191	2	21	255	0	192
22	0	22	255	66	0	107	1	22	255	189	66	192	2	22	255	0	189
23	0	23	255	69	0	108	1	23	255	186	69	193	2	23	255	0	186
24	0	24	255	72	0	109	1	24	255	183	72	194	2	24	255	0	183
25	0	25	255	75	0	110	1	25	255	180	75	195	2	25	255	0	180
26	0	26	255	78	0	111	1	26	255	177	78	196	2	26	255	0	177
27	0	27	255	81	0	112	1	27	255	174	81	197	2	27	255	0	174
28	0	28	255	84	0	113	1	28	255	171	84	198	2	28	255	0	171
29	0	29	255	87	0	114	1	29	255	168	87	199	2	29	255	0	168
30	0	30	255	90	0	115	1	30	255	165	90	200	2	30	255	0	165
31	0	31	255	93	0	116	1	31	255	162	93	201	2	31	255	0	162
32	0	32	255	96	0	117	1	32	255	159	96	202	2	32	255	0	159
33	0	33	255	99	0	118	1	33	255	156	99	203	2	33	255	0	156
34	0	34	255	102	0	119	1	34	255	153	102	204	2	34	255	0	153
35	0	35	255	105	0	120	1	35	255	150	105	205	2	35	255	0	150
36	0	36	255	108	0	121	1	36	255	147	108	206	2	36	255	0	147
37	0	37	255	111	0	122	1	37	255	144	111	207	2	37	255	0	144
38	0	38	255	114	0	123	1	38	255	141	114	208	2	38	255	0	141
39	0	39	255	117	0	124	1	39	255	138	117	209	2	39	255	0	138
40	0	40	255	120	0	125	1	40	255	135	120	210	2	40	255	0	135
41	0	41	255	123	0	126	1	41	255	132	123	211	2	41	255	0	132
42	0	42	255	126	0	127	1	42	255	129	126	212	2	42	255	0	129
43	0	43	255	129	0	128	1	43	255	126	129	213	2	43	255	0	126
44	0	44	255	132	0	129	1	44	255	123	132	214	2	44	255	0	123
45	0	45	255	135	0	130	1	45	255	120	135	215	2	45	255	0	120
46	0	46	255	138	0	131	1	46	255	117	138	216	2	46	255	0	117
47	0	47	255	141	0	132	1	47	255	114	141	217	2	47	255	0	114
48	0	48	255	144	0	133	1	48	255	111	144	218	2	48	255	0	111
49	0	49	255	147	0	134	1	49	255	108	147	219	2	49	255	0	108
50	0	50	255	150	0	135	1	50	255	105	150	220	2	50	255	0	105
51	0	51	255	153	0	136	1	51	255	102	153	221	2	51	255	0	102
52	0	52	255	156	0	137	1	52	255	99	156	222	2	52	255	0	99
53	0	53	255	159	0	138	1	53	255	96	159	223	2	53	255	0	96
54	0	54	255	162	0	139	1	54	255	93	162	224	2	54	255	0	93
55	0	55	255	165	0	140	1	55	255	90	165	225	2	55	255	0	90
56	0	56	255	168	0	141	1	56	255	87	168	226	2	56	255	0	87
57	0	57	255	171	0	142	1	57	255	84	171	227	2	57	255	0	84
58	0	58	255	174	0	143	1	58	255	81	174	228	2	58	255	0	81
59	0	59	255	177	0	144	1	59	255	78	177	229	2	59	255	0	78
60	0	60	255	180	0	145	1	60	255	75	180	230	2	60	255	0	75
61	0	61	255	183	0	146	1	61	255	72	183	231	2	61	255	0	72
62	0	62	255	186	0	147	1	62	255	69	186	232	2	62	255	0	69
63	0	63	255	189	0	148	1	63	255	66	189	233	2	63	255	0	66
64	0	64	255	192	0	149	1	64	255	63	192	234	2	64	255	0	63
65	0	65	255	195	0	150	1	65	255	60	195	235	2	65	255	0	60
66	0	66	255	198	0	151	1	66	255	57	198	236	2	66	255	0	57
67	0	67	255	201	0	152	1	67	255	54	201	237	2	67	255	0	54
68	0	68	255	204	0	153	1	68	255	51	204	238	2	68	255	0	51
69	0	69	255	207	0	154	1	69	255	48	207	239	2	69	255	0	48
70	0	70	255	210	0	155	1	70	255	45	210	240	2	70	255	0	45
71	0	71	255	213	0	156	1	71	255	42	213	241	2	71	255	0	42
72	0	72	255	216	0	157	1	72	255	39	216	242	2	72	255	0	39
73	0	73	255	219	0	158	1	73	255	36	219	243	2	73	255	0	36
74	0	74	255	222	0	159	1	74	255	33	222	244	2	74	255	0	33
75	0	75	255	225	0	160	1	75	255	30	225	245	2	75	255	0	30
76	0	76	255	228	0	161	1	76	255	27	228	246	2	76	255	0	27
77	0	77	255	231	0	162	1	77	255	24	231	247	2	77	255	0	24
78	0	78	255	234	0	163	1	78	255	21	234	248	2	78	255	0	21
79	0	79	255	237	0	164	1	79	255	18	237	249	2	79	255	0	18
80	0	80	255	240	0	165	1	80	255	15	240	250	2	80	255	0	15
81	0	81	255	243	0	166	1	81	255	12	243	251	2	81	255	0	12
82	0	82	255	246	0	167	1	82	255	9	246	252	2	82	255	0	9
83	0	83	255	249	0	168	1	83	255	6	249	253	2	83	255	0	6
84	0	84	255	252	0	169	1	84	255	3	252	254	2	84	255	0	3
85	1	0	255	255	0	170	2	0	255	0	255	255	0	0	255	0	0

Αυτό ακριβώς υλοποιεί και πετυχαίνει το παρακάτω υποπρόγραμμα **xrwmaRYMR**, το οποίο προστίθεται ακριβώς κάτω από το υπάρχον υποπρόγραμμα **xrwma** στον κώδικα 7Kallitexnies26.py.

```
def xrwma(a):
    '''Dexetai enan akeraio arithmo a kai dhmiourgei thn antistoixh apoxrwsh.
    Geitonikoi arithmoi exoyn syggenikes apoxrwses me poreia apoxrwsewn
    kokkino, kitrino, prasino, galazio, mple, matzenta, kokkino ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 43 #To 43 xrhsimopoeitai dioti moirazei sto peripoy
    ypD = a % 43 #thn perioxh 0 ews 255 se 6 isa merh.
    if akM == 0: xrwmatismos = (255, ypD*255/43, 0)
    elif akM == 1: xrwmatismos = (255-ypD*255/43, 255, 0)
    elif akM == 2: xrwmatismos = (0, 255, ypD*255/43)
    elif akM == 3: xrwmatismos = (0, 255-ypD*255/43, 255)
    elif akM == 4: xrwmatismos = (ypD*255/43, 0, 255)
    elif akM == 5: xrwmatismos = (255, 0, 255-ypD*255/43)
    return xrwmatismos

def xrwmaRYMR(a):
    '''Dexetai enan akeraio arithmo a kai dhmiourgei thn antistoixh apoxrwsh.
    me poreia apoxrwsewn kokkino, kitrino, matzenta, kokkino ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 85 #To 85 xrhsimopoeitai dioti moirazei sto peripoy
    ypD = a % 85 #thn perioxh 0 ews 255 se 3 isa merh.
    if akM == 0: xrwmatismos = (255, ypD*255/85, 0)
    elif akM == 1: xrwmatismos = (255, 255-ypD*255/85, ypD*255/85)
    elif akM == 2: xrwmatismos = (255, 0, 255-ypD*255/85)
    return xrwmatismos
```

Επίσης, παρακάτω στην εντολή σχεδίασης γραμμής **pygame.draw.line**, αλλάζουμε το όρισμα χρωματισμού από **xrwma(c)** σε **xrwmaRYMR(c)**. Παρατηρεί κανείς ότι έχουν αντικατασταθεί τα μακροσκελή ονόματα μεταβλητών **akeraioMeros** και **γρολοιροDiaireshs** στα πιο σύντομα **akM** και **ypD**. Αποθηκεύουμε ως 7Kallitexnies27.py και τρέχουμε.



Η ονομασία του υποπρογράμματος **xrwmaRYMR** δόθηκε έτσι ώστε να αντικατοπτρίζει την πορεία των χρωματισμών στο χρόνο (R για RED-κόκκινο, Y για YELLOW-κίτρινο κτλ.). Καθώς τώρα υπάρχουν τρεις περιοχές μετάβασης (κόκκινο σε κίτρινο, κίτρινο σε ματζέντα, ματζέντα σε κόκκινο), συνιστάται να έχουμε 85 βήματα χρωματισμού σε κάθε μετάβαση, κάτι που προκύπτει από τη διαίρεση 255 διά 3. Γι' αυτό το λόγο βλέπουμε συχνά το 85 ως διαιρέτη στο υποπρόγραμμα **xrwmaRYMR(c)**, σε αντίθεση με το προηγούμενο υποπρόγραμμα **xrwma(c)** όπου ο διαιρέτης ήταν το 43 καθώς επιθυμούσαμε έξι περιοχές μετάβασης (το 43 μοιράζει το 255 περίπου σε έξι μέρη).

ΚΕΦΑΛΑΙΟ 28, Έγχρωμος κύκλος 03

Κατανοώντας τη λογική που διέπει τα δύο προηγούμενα υποπρογράμματα, μπορεί κανείς εύκολα πλέον να γράψει υποπρογράμματα που να υλοποιούν διαφορετικούς χρωματισμούς, π.χ. κοντά στο πράσινο ή κοντά στο μπλε.

		R	G	B
G	πράσινο	0	255	0
C	γαλάζιο	0	255	255
Y	κίτρινο	255	255	0
G	πράσινο	0	255	0

		R	G	B
B	μπλε	0	0	255
M	ματζέντα	255	0	255
C	γαλάζιο	0	255	255
B	μπλε	0	0	255

Οι παραπάνω πίνακες υλοποιούνται με τα παρακάτω υποπρογράμματα, τα οποία τα προσθέτουμε λοιπόν και αυτά στο προηγούμενο πρόγραμμα.

```
def xrwmaGCG(a):
    '''Dexetai enan akeraio arithmo a kai dhmiourgei thn antistoixh apoxrwsh
    me poreia apoxrwsewn prasino, kyano, kitrino, prasino ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 85 #To 85 xrhsimopieitai dioti moirazei sto peripoy
    ypD = a % 85 #thn perioxh 0 ews 255 se 3 isa merh.
    if akM == 0: xrwmatismos = (0, 255, ypD*255/85)
    elif akM == 1: xrwmatismos = (ypD*255/85, 255, 255-ypD*255/85)
    elif akM == 2: xrwmatismos = (255-ypD*255/85, 255, 0)
    return xrwmatismos

def xrwmaBMCB(a):
    '''Dexetai enan akeraio arithmo a kai dhmiourgei thn antistoixh apoxrwsh
    me poreia apoxrwsewn mple, matzenta, kyano, mple ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 85 #To 85 xrhsimopieitai dioti moirazei sto peripoy
    ypD = a % 85 #thn perioxh 0 ews 255 se 3 isa merh.
    if akM == 0: xrwmatismos = (ypD*255/85, 0, 255)
    elif akM == 1: xrwmatismos = (255-ypD*255/85, ypD*255/85, 255)
    elif akM == 2: xrwmatismos = (0, 255-ypD*255/85, 255)
    return xrwmatismos
```

Όπως εξηγήθηκε και προηγουμένως, οι ονομασίες των υποπρογραμμάτων δίνονται έτσι ώστε να αντικατοπτρίζουν την πορεία των χρωματισμών στο χρόνο. Βέβαια, για να δοκιμάσει κανείς το αποτέλεσμα, απαιτείται να αλλάξει το όρισμα χρωματισμού στην εντολή σχεδίασης γραμμής `pygame.draw.line`, είτε σε `xrwmaGCG(c)` είτε σε `xrwmaBMCB(c)`. Αποθηκεύουμε ως `7Kallitexnies28.py` και τρέχουμε.



ΚΕΦΑΛΑΙΟ 29, Έγχρωμος κύκλος 04

Στην ίδια λογική που διέπει τα προηγούμενα υποπρογράμματα, κινούνται και τα επόμενα δύο υποπρογράμματα που υλοποιούν διαφορετικούς χρωματισμούς, σύμφωνα με τους παρακάτω πίνακες.

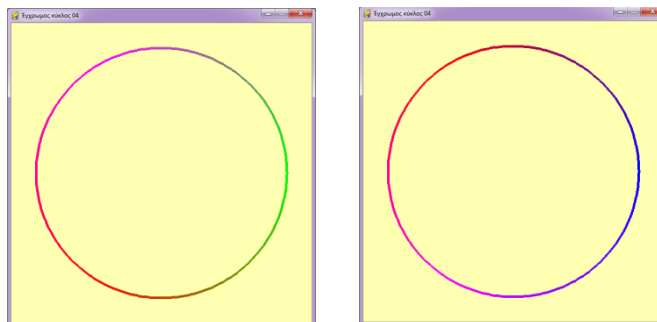
		R	G	B
G	πράσινο	0	255	0
M	ματζέντα	255	0	255
R	κόκκινο	255	0	0
G	πράσινο	0	255	0

		R	G	B
B	μπλε	0	0	255
R	κόκκινο	255	0	0
M	ματζέντα	255	0	255
B	μπλε	0	0	255

```
def xrwmaMRG(a):
    '''Dexetai enan akeraio arithmo a kai dhmiourgei thn antistoixh apoxrwsh
    me poreia apoxrwsewn prasino, matzenta, kokkino, prasino ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 85 #To 85 xrhsimopieitai dioti moirazei sto peripoy
    ypD = a % 85 #thn perioxh 0 ews 255 se 3 isa merh.
    if akM == 0: xrwmatismos = (ypD*255/85, 255-ypD*255/85, ypD*255/85)
    elif akM == 1: xrwmatismos = (255, 0, 255-ypD*255/85)
    elif akM == 2: xrwmatismos = (255-ypD*255/85, ypD*255/85, 0)
    return xrwmatismos

def xrwmaBRMB(a):
    '''Dexetai enan akeraio arithmo a kai dhmiourgei thn antistoixh apoxrwsh
    me poreia apoxrwsewn mple, kokkino, matzenta, mple ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 85 #To 85 xrhsimopieitai dioti moirazei sto peripoy
    ypD = a % 85 #thn perioxh 0 ews 255 se 3 isa merh.
    if akM == 0: xrwmatismos = (ypD*255/85, 0, 255-ypD*255/85)
    elif akM == 1: xrwmatismos = (255, 0, ypD*255/85)
    elif akM == 2: xrwmatismos = (255-ypD*255/85, 0, 255)
    return xrwmatismos
```

Προσθέτουμε και αυτά τα υποπρογράμματα, αποθηκεύουμε ως 7Kallitexnies29.py και κάνουμε τις δοκιμές μας αλλάζοντας το όρισμα χρωματισμού στην `pygame.draw.line`.



ΚΕΦΑΛΑΙΟ 30, Έγχρωμος κύκλος 05 με εντολή `import`

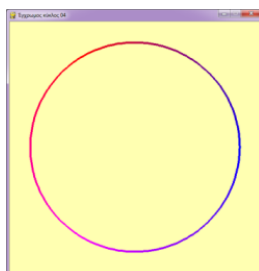
Το πρόγραμμά μας έχει γίνει αρκετά μεγάλο και καλό είναι να το σπάσουμε σε δύο τμήματα. Το ένα θα αποτελείται από τα υποπρογράμματα χρωματισμών και το άλλο θα αποτελείται από τις υπόλοιπες εντολές. Δημιουργούμε λοιπόν ένα ξεχωριστό αρχείο `χρωματισμοί.py` το οποίο περιέχει μόνο τα έξι υποπρογράμματα. Πώς θα έχουμε πρόσβαση σε αυτό; Αρκεί γι' αυτό η εντολή `from χρωματισμοί import *` στη θέση που προηγουμένως υπήρχαν τα υποπρογράμματα. Έτσι, το πρόγραμμά μας παίρνει την εξής μορφή:

```
7Kallitexnies30.py - C:/Python33/7Kallitexnies30.py
File Edit Format Run Options Windows Help
import pygame, sys, os, time; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'

from χρωματισμοί import *
# To αρχείο χρωματισμοί.py περιέχει τα παρακάτω 6 υποπρογράμματα
# xrwma(a), xrwmaRYMR(a), xrwmaGCGY(a), xrwmaBMCB(a), xrwmaGMRG(a), xrwmaBRMB(a)

SIZE=600 #Megethos parathyroy
R=250 #Aktina kykloy
N=255 #Plnthos tmhmatwn se kyklo
PG=5 #Pachos grammhs
DT=0.01 #Xronos diakophs se deyteerolepta

pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Έγχρωμος κύκλος 05')
epif.fill((255,255,177)) #Xrwmatismos anoixtoxrwmos mpez
i=0 #Metrhths
c=1 #Akeraios arithmos poy tha ayksanetai synexeia kai tha metatrepetai se xrwma
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    if i<N:
        pygame.draw.line(epif,xrwmaGMRG(c), (SIZE/2+R*cos(i*2*pi/N),SIZE/2-R*sin(i*2*pi/N)),
            (SIZE/2+R*cos((i+1)*2*pi/N),SIZE/2-R*sin((i+1)*2*pi/N)),PG)
        time.sleep(DT); pygame.display.update(); c=c+1
        i=i+1
    pygame.display.update()
```

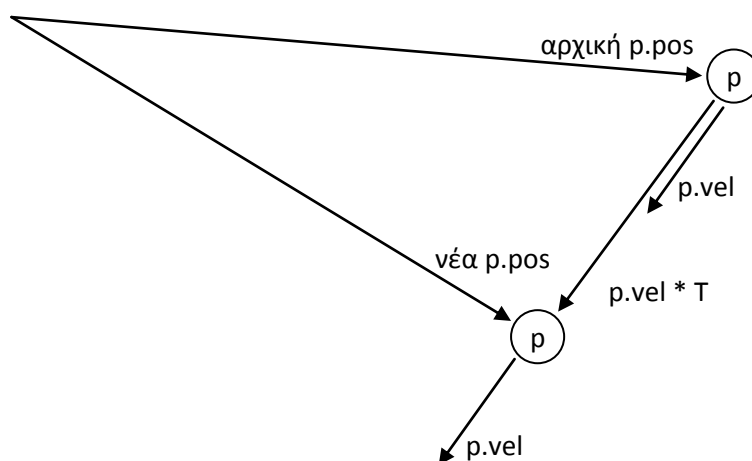


Αποθηκεύουμε ως 7Kallitexnies30.py και εκτελούμε. Δεν πρέπει να παρατηρήσουμε καμία διαφορά στην εκτέλεση σε σχέση με το προηγούμενο πρόγραμμα.

Για το αρχείο με τα υποπρόγραμμα σημειώνεται πρώτον ότι η ονομασία του δεν επιτρέπεται να εκκινεί με αριθμό (θα ήταν λάθος π.χ. η ονομασία `7xrwmatismoi.py`) και δεύτερον ότι πρέπει να τοποθετείται στο φάκελο της Python.

ΚΕΦΑΛΑΙΟ 31, Ευθύγραμμες κινήσεις με διανύσματα

Ξεκινώντας από αυτό το κεφάλαιο, θα παρουσιάσουμε μια εναλλακτική προσέγγιση σχεδίασης, η οποία παρουσιάζει απίστευτη δυναμική. Πλέον, θα θεωρούμε κινητά αντικείμενα στον καμβά, τα οποία αφήνουν το έγχρωμο στίγμα τους καθώς προχωρούν. Η μαθηματική βάση στην οποία στηρίζονται αυτά τα κινητά αντικείμενα είναι τα διανύσματα. Κάθε αντικείμενο, λοιπόν, θα έχει το δικό του διάνυσμα θέσης (ουσιαστικά οι συντεταγμένες του), αλλά θα έχει και το δικό του διάνυσμα ταχύτητας (ουσιαστικά η φορά κίνησής του και το μέτρο της ταχύτητας) κάτι που αναγκαστικά θα προσδιορίζει τη θέση του κινητού την επόμενη χρονική στιγμή. Έτσι, κρίνεται απαραίτητο να δοθούν από την αρχή κάποιες σχετικές μαθηματικές γνώσεις.



Θεωρούμε το κινητό αντικείμενο p (από το αρχικό λατινικό γράμμα της λέξης πλανήτης). Οι συντεταγμένες του είναι το λεγόμενο διάνυσμα θέσης του $p.pos$ (pos από position που σημαίνει θέση), το οποίο μπορεί κανείς να το θεωρήσει και ως ένα ζεύγος (x,y) . Η ταχύτητά του είναι το λεγόμενο διάνυσμα ταχύτητας $p.vel$ (vel από velocity που σημαίνει ταχύτητα), το οποίο μπορεί κανείς να το φανταστεί ως ένα διάνυσμα που εκπορεύεται με συγκεκριμένη φορά από το αντικείμενο p . Με βάση αυτά τα δύο στοιχεία, πώς άραγε υπολογίζεται η νέα θέση του αντικειμένου p , ή με άλλα λόγια το νέο διάνυσμα θέσης του (το νέο $p.pos$) έπειτα από χρόνο T ;

Από τη Φυσική γνωρίζουμε ότι η ταχύτητα προκαλεί μια αλλαγή στη θέση ενός κινητού. Ισχύει ο τύπος της ταχύτητας $u=s/t$, ή με άλλα λόγια $s=u*t$. Στο σχήμα μας, λοιπόν, η ταχύτητα $p.vel$ προκαλεί μια μετατόπιση $p.vel*T$, η οποία ουσιαστικά είναι ένα διάνυσμα θέσης. Προσθέτοντας το αρχικό διάνυσμα θέσης $p.pos$ με το διάνυσμα θέσης που προκλήθηκε από την ταχύτητα, παίρνουμε το τελικό διάνυσμα θέσης της επόμενης χρονικής στιγμής. Συνολικά: $\text{νέο } p.pos = \text{αρχικό } p.pos + p.vel*T$.

Όταν τα διανύσματα εκφράζονται ως ζεύγη αριθμών, το άθροισμά τους υπολογίζεται αθροίζοντας τις επιμέρους συντεταγμένες, ενώ το γινόμενο διανύσματος με αριθμό υπολογίζεται πολλαπλασιάζοντας κάθε μια από τις δύο συντεταγμένες με τον αριθμό.

$$(\alpha,\beta)+(\gamma,\delta)=(\alpha+\gamma,\beta+\delta) \text{ και } (\alpha,\beta)*T=(\alpha*T, \beta*T)$$

Είμαστε έτοιμοι να δοκιμάσουμε το πρώτο σχετικό πρόγραμμα. Γράφουμε τον παρακάτω κώδικα 7Kallitexnies31.py και τον εκτελούμε.

```

74 Kallitexnies31.py - C:\Python33\Kallitexnies31.py
File Edit Format Run Options Windows Help
import pygame, sys, os, time; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'

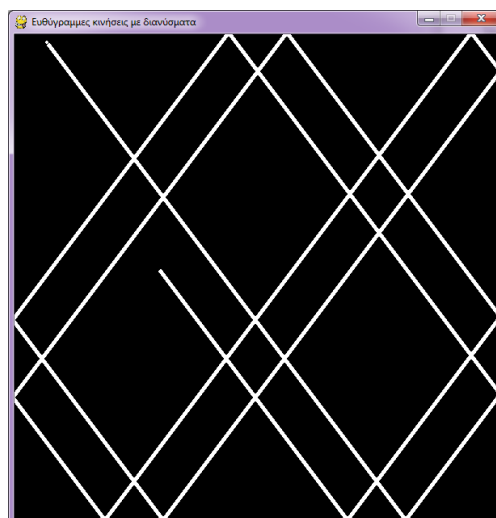
class Object(object):
    ''' Genika antikeimena'''
    pass
class V2D():
    '''Dianysmata 2 diastasewn'''
    def __init__(self, data):
        self.data = data
    def __repr__(self):
        return repr(self.data)
    def __setitem__(self, key, item):
        self.data[key] = item
    def __getitem__(self, key):
        return self.data[key]
    def __add__(self, other):
        return V2D([self.data[0] + other.data[0], self.data[1] + other.data[1]])
    def __sub__(self, other):
        return V2D([self.data[0] - other.data[0], self.data[1] - other.data[1]])
    def __mul__(self, other):
        return V2D([self.data[0] * other, self.data[1] * other])

SIZE=600 #Megethos parathyroy
T=1      #Xronos metaksy dyo diadoxikwn thesewn

pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Ευθύγραμμες κινήσεις με διανύσματα')

p=Object() #Kinhto antikeimeno me thesh kai taxythta
p.pos = V2D([40,10]) #Arxikh thesh kinhtoy
p.vel = V2D([3,4])   #Arxikh taxythta kinhtoy
pygame.draw.circle(epif, (255,255,255), (p.pos[0],p.pos[1]),2)
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    p.pos=p.pos+p.vel*T #Υπολογισμος neas theshs
    print(p.pos)
    pygame.draw.circle(epif, (255,255,255), (int(p.pos[0]),int(p.pos[1])),3)
    if p.pos[0]>SIZE or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
    elif p.pos[1]>SIZE or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
    time.sleep(0.01) #Elaxisth xronikh diakoph
    pygame.display.update()

```



Παρατηρούμε ένα κινητό να τρέχει βολίδα αφήνοντας πίσω το άσπρο στίγμα του. Ας δώσουμε τις απαραίτητες εξηγήσεις στον κώδικα.

Τα τμήματα `class Object()` και `class V2D()` χρησιμεύουν ακριβώς σε αυτήν την ιδέα που ειπώθηκε προηγουμένως, δηλαδή τη θεώρηση αντικειμένων που κινούνται σε δύο διαστάσεις με κάποια ταχύτητα. Τα υποπρογράμματα που υπάρχουν σε εσοχή στο `class V2D()`, αφορούν πράξεις που εφαρμόζονται στα αντικείμενα τύπου διανύσματος. Για παράδειγμα, στα υποπρογράμματα `__add__`, `__sub__`, και `__mul__` δηλώνεται η συμπεριφορά που θα έχουν τα σύμβολα `+`, `-` και `*` σε παραστάσεις με σχετικά αντικείμενα. Δεν θα επεκταθούμε σε περισσότερες επεξηγήσεις εδώ, και αυτό θα γίνει αργότερα όταν κριθεί σκόπιμο.

Αυτό το πρόγραμμα διαθέτει μόνο ένα αντικείμενο. Αυτό δημιουργείται με την εντολή `p=Object()`. Όμως, αυτό το αντικείμενο έχει δύο ιδιότητες (τα διανύσματα θέσης και ταχύτητας) που και αυτές θεωρούνται αντικείμενα. Αυτές οι ιδιότητες ορίζονται με τις δύο επόμενες εντολές `p.pos = V2D([40,10])` και `p.vel = V2D([3,4])`. Με την επόμενη εντολή `pygame.draw.circle` σχεδιάζεται ένα άσπρο στίγμα στην αρχική θέση του κινητού, και γι' αυτό χρησιμοποιείται η παράσταση `(p.pos[0],p.pos[1])` ως τρίτο όρισμα. Υπενθυμίζεται ότι σε αυτό το όρισμα του κέντρου του κύκλου απαιτούνται δύο ακέραιοι. Στην αρχική θέση δεν υπάρχει πρόβλημα, καθώς πρόκειται για τους ακέραιους αριθμούς 40 και 10. Όμως παρακάτω παρατηρούμε την ελαφρά διαφορετική παράσταση `(int(p.pos[0]),int(p.pos[1]))` και αυτό γίνεται διότι απαιτείται η συνάρτηση `int` που μετατρέπει πιθανές δεκαδικές τιμές σε ακέραιες.

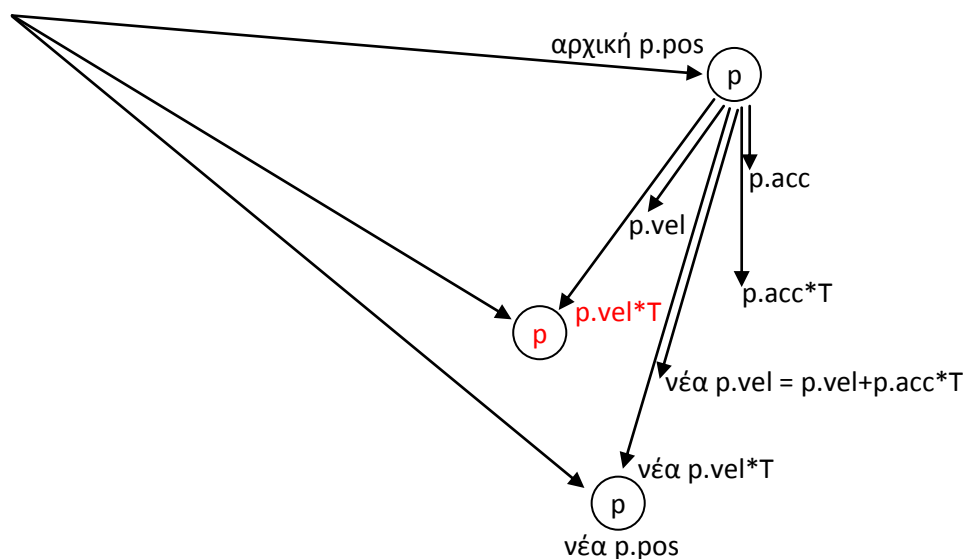
Εντός του `Loop`, μετά το σύνθημα και γνωστό τμήμα `for event...`, παρατηρούμε την εντολή `p.pos=p.pos+p.vel*T`. Εδώ ακριβώς υλοποιείται αυτό που επεξηγήθηκε στην αρχή του κεφαλαίου. Προστίθεται το αρχικό διάνυσμα θέσης `p.pos` με το διάνυσμα θέσης που προκλήθηκε από την ταχύτητα, και προκύπτει το διάνυσμα θέσης `p.pos` της επόμενης χρονικής στιγμής. Η εντολή `print(p.pos)` δεν είναι απαραίτητη και μπορεί και να απουσιάζει, όμως όταν υπάρχει έχει ως αποτέλεσμα να εκτυπώνει στο κέλυφος της Python το τρέχον αντικείμενο `p.pos`, δηλαδή τις συντεταγμένες του αντικειμένου. Η επόμενη εντολή `pygame.draw.circle` σχεδιάζει ένα μικρό κύκλο στη θέση `p.pos`, ώστε να σχεδιάζεται τελικά η πορεία του κινητού. Η επόμενη εντολή `if p.pos[0]>SIZE or p.pos[0]<0:` ελέγχει εάν το κινητό έχει βρεθεί εκτός παραθύρου από τα δεξιά ή από τα αριστερά, ενώ η συνέχεια της εντολής αυτής `elif p.pos[1]>SIZE or p.pos[1]<0:` ελέγχει εάν το κινητό έχει βρεθεί εκτός παραθύρου από κάτω ή από επάνω. Το `p.pos[0]` αναφέρεται στη συντεταγμένη του άξονα x , ενώ το `p.pos[1]` αναφέρεται στη συντεταγμένη του άξονα y . Παρατηρεί κανείς ότι στις πρώτες δύο περιπτώσεις αρκεί να αλλάξουμε πρόσημο στην πρώτη συντεταγμένη της ταχύτητας με `p.vel[0]=p.vel[0]*(-1)`, ενώ στις δύο τελευταίες περιπτώσεις αρκεί να αλλάξουμε πρόσημο στην δεύτερη συντεταγμένη της ταχύτητας με `p.vel[1]=p.vel[1]*(-1)`.

Η εντολή `time.sleep(0.01)` είναι ο χρόνος σε δευτερόλεπτα που μεσολαβεί μεταξύ των διαρκών υπολογισμών των θέσεων του κινητού.

ΚΕΦΑΛΑΙΟ 32, Καμπύλες κινήσεις με διανύσματα

Στο προηγούμενο κεφάλαιο το κινητό μας κινείται μόνο ευθύγραμμα ενώ ο στόχος σε αυτό το κεφάλαιο είναι να επιτύχουμε καμπύλη κίνηση. Ένας ενδεδειγμένος τρόπος είναι να χρησιμοποιηθεί η έννοια της επιτάχυνσης, η οποία μπορεί να αλλάξει δραστικά τη φορά της ταχύτητας του κινητού και άρα να προκαλέσει καμπύλες πορείες.

Πριν την προγραμματιστική υλοποίηση του ζητήματος αυτού, καλό είναι να προηγηθούν κάποιες θεωρητικές επεξηγήσεις από τα Μαθηματικά και τη Φυσική.



Στο παραπάνω σχήμα βλέπουμε με κόκκινο την επόμενη θέση του κινητού όταν δεν ασκείται επάνω του επιτάχυνση και οι εξηγήσεις έχουν δοθεί στο προηγούμενο κεφάλαιο. Όταν όμως υπάρχει και επιτάχυνση $p.acc$ (acc από το $acceleration$ που σημαίνει επιτάχυνση), τότε τα πράγματα γίνονται ελαφρώς πιο περίπλοκα. Διότι πλέον η ταχύτητα $p.vel$ δεν θα παραμένει σταθερή αλλά και αυτή θα αλλάζει σε κάθε βήμα. Δοθέντων της θέσης $p.pos$, της ταχύτητας $p.vel$ και της επιτάχυνσης $p.acc$ ενός κινητού αντικειμένου, πώς άραγε υπολογίζονται η νέα ταχύτητα και η νέα θέση του έπειτα από χρόνο T ;

Από τη Φυσική γνωρίζουμε ότι η επιτάχυνση προκαλεί αλλαγή στην ταχύτητα ενός κινητού και ισχύει ο τύπος της επιτάχυνσης $a = u/t$, ή με άλλα λόγια $u = a * t$. Αυτή η ταχύτητα που προκλήθηκε από την επιτάχυνση πρέπει να προστεθεί διανυσματικά στην αρχική ταχύτητα του κινητού. Αντίστοιχα στο σχήμα μας, η επιτάχυνση $p.acc$ προκαλεί την ταχύτητα $p.acc * T$, η οποία προστίθεται στην αρχική ταχύτητα $p.vel$ και προκύπτει η νέα $p.vel$ που ισούται τελικά με $p.vel + p.acc * T$. Αυτή η νέα $p.vel$ προκαλεί μια μετατόπιση $p.vel * T$, η οποία ουσιαστικά είναι ένα διάνυσμα θέσης. Προσθέτοντας το αρχικό διάνυσμα θέσης $p.pos$ με το διάνυσμα θέσης που προκλήθηκε από την ταχύτητα, παίρνουμε το τελικό διάνυσμα θέσης της επόμενης χρονικής στιγμής. Συνολικά:

$$\text{νέα } p.vel = \text{αρχική } p.vel + p.acc * T$$

$$\text{νέα } p.pos = \text{αρχική } p.pos + (p.vel + p.acc * T) * T.$$

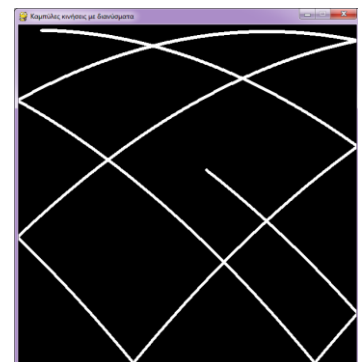
Όλα τα παραπάνω θεωρητικά, μένει τώρα να υλοποιηθούν και προγραμματιστικά. Στο πρόγραμμα αυτού του κεφαλαίου η επιτάχυνση θεωρείται σταθερή, ενώ σε επόμενα κεφάλαια, όπου θα προσπαθήσουμε να προσομοιώσουμε βαρύτητα π.χ. με κινήσεις πλανητών γύρω από ήλιους, η επιτάχυνση θα αλλάζει σε κάθε βήμα.

Γράφουμε τον παρακάτω κώδικα `7Kallitexnies32.py` και τον εκτελούμε.

```
74 Kallitexies32.py - C:/Python33/7Kallitexies32.py
File Edit Format Run Options Windows Help
import pygame, sys, os, time; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'

class Object(object):
    '''Antikeimena genika'''
    pass
class V2D():
    '''Antikeimena dianysmata dyo diastasewn'''
    def __init__(self, data):
        self.data = data
    def __repr__(self):
        return repr(self.data)
    def __setitem__(self, key, item):
        self.data[key] = item
    def __getitem__(self, key):
        return self.data[key]
    def __add__(self, other):
        return V2D([self.data[0] + other.data[0], self.data[1] + other.data[1]])
    def __sub__(self, other):
        return V2D([self.data[0] - other.data[0], self.data[1] - other.data[1]])
    def __mul__(self, other):
        return V2D([self.data[0] * other, self.data[1] * other])

SIZE=600 #Megethos parathyroy
T=0.01 #Xronos metaksy dyo diadoxikwn thesewn
pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Καμπύλες κινήσεις με διανύσματα')
p=Object() #Kinhtho antikeimeno me thesh, taxythta kai epitaxyynsh
p.pos = V2D([40,10]) #Arxikh thesh kinhtoy
p.vel = V2D([200,3]) #Arxikh taxythta kinhtoy
p.acc = V2D([0,50]) #Epitaxyynsh
pygame.draw.circle(epif, (255,255,255), (p.pos[0],p.pos[1]),2)
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    p.vel=p.vel+p.acc*T #Υπολογισμος neas taxythtas
    p.pos=p.pos+p.vel*T #Υπολογισμος neas theshs
    pygame.draw.circle(epif, (255,255,255), (int(p.pos[0]),int(p.pos[1])),3)
    if p.pos[0]>SIZE or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
    elif p.pos[1]>SIZE or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
    time.sleep(0.01) #Elaxisth xronikh diakroph
    pygame.display.update()
```



Πράγματι, επετεύχθη η καμπύλη πορεία και παρατηρεί κανείς μια αίσθηση έλξης του κινητού αντικειμένου προς τα κάτω.

Παρατηρεί κανείς ότι σε σχέση με το προηγούμενο πρόγραμμα έχει προστεθεί η εντολή `p.acc = V2D([0,50])`. Αυτό υποδηλώνει την ιδιότητα επιτάχυνσης του αντικειμένου `p`, η οποία όπως φαίνεται από τους αριθμούς έχει φορά προς τα κάτω. Αυτή η ιδιότητα, σε αντίθεση με τις ιδιότητες θέσης και ταχύτητας, ποτέ δεν αλλάζει. Εν συνεχεία, μέσα στο άεναιο loop οι εντολές `p.vel=p.vel+p.acc*T` και `p.pos=p.pos+p.vel*T` υπολογίζουν ακατάπαυστα την επόμενη θέση και ταχύτητα του κινητού αντικειμένου, σύμφωνα με όσα επεξηγήθηκαν στην αρχή. Ακολουθεί η σχεδίαση του τρέχοντος στίγματος με την εντολή `pygame.draw.circle`. Αμέσως μετά η εντολή `if` που επεξηγήθηκε και στο προηγούμενο κεφάλαιο ελέγχει την περίπτωση που η θέση του κινητού βρεθεί εκτός παραθύρου και πράττει ανάλογα. Μετά η `time.sleep(0.01)` προκαλεί μια μικρή χρονοκαυστέρηση σε κάθε βήμα. Και τέλος η `pygame.display.update()` ανανεώνει όλα τα δεδομένα της οθόνης.

ΚΕΦΑΛΑΙΟ 33, Κίνηση πλανήτη γύρω από ήλιο

Για λόγους μαζέματος του κώδικα, ανοίγουμε το αρχείο `xrwmatismoι.py` και προσθέτουμε προς το τέλος του το τμήμα κώδικα που δηλώνονται τα class. Καθώς τώρα το αρχείο περιέχει και υποπρογράμματα χρωματισμών και δηλώσεις class, ας το μετονομάσουμε καλύτερα σε `defClass.py`, ώστε η ονομασία να αντανakλά καλύτερα το περιεχόμενο.

Το `defClass.py` πρέπει τώρα να έχει την εξής μορφή προς το τέλος του:

```

elif akM == 2: xrwmatismos = (255-ypD*255/85, ypD*255/85, 0)
return xrwmatismos

def xrwmaBRMB(a):
    '''Dexetai enan akeraio arithmo a kai dhmioyrgei thn antistoixh apoxrwsh
    me poreia apoxrwsewn mple, kokkino, matzenta, mple ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 85 #To 85 xrhsimopoieitai dioti moirazei sto peripoy
    ypD = a % 85 #thn perioxh 0 ews 255 se 3 isa merh.
    if akM == 0: xrwmatismos = (ypD*255/85, 0, 255-ypD*255/85)
    elif akM == 1: xrwmatismos = (255, 0, ypD*255/85)
    elif akM == 2: xrwmatismos = (255-ypD*255/85, 0, 255)
    return xrwmatismos

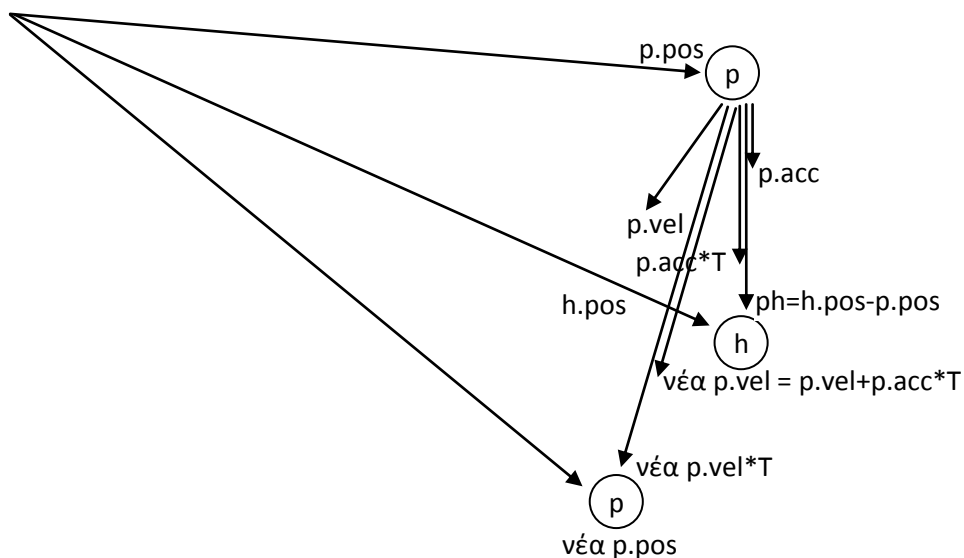
class Object(object):
    '''Antikeimena genika'''
    pass

class V2D():
    '''Antikeimena dianysmata dyo diastasewn'''
    def __init__(self, data):
        self.data = data
    def __repr__(self):
        return repr(self.data)
    def __setitem__(self, key, item):
        self.data[key] = item
    def __getitem__(self, key):
        return self.data[key]
    def __add__(self, other):
        return V2D([self.data[0] + other.data[0], self.data[1] + other.data[1]])
    def __sub__(self, other):
        return V2D([self.data[0] - other.data[0], self.data[1] - other.data[1]])
    def __mul__(self, other):
        return V2D([self.data[0] * other, self.data[1] * other])

```

Πλέον, με εντολή `import` θα ενσωματώνουμε αυτό το αρχείο στο κύριο πρόγραμμα.

Σε αυτό το κεφάλαιο θα επιχειρήσουμε να προσομοιώσουμε καμπύλη κίνηση πλανήτη γύρω από ήλιο. Και πάλι κρίνεται απαραίτητο να δοθούν στην αρχή κάποιες θεωρητικές γνώσεις, και πάλι ίσως ένα σχήμα βοηθήσει στην κατανόηση.

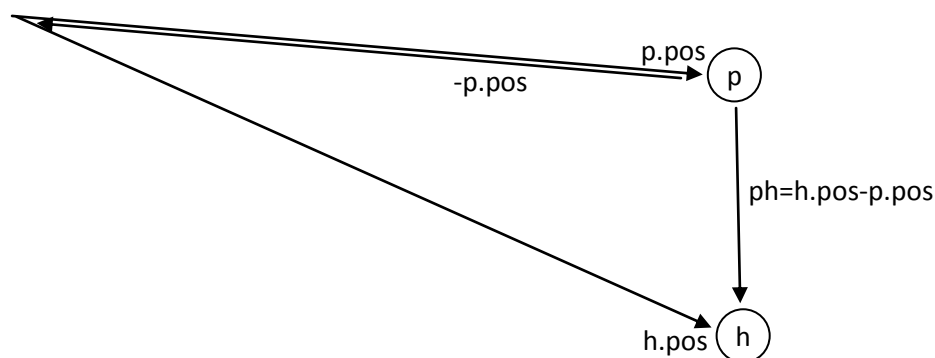


Η βασική διαφορά με το προηγούμενο παράδειγμα έγκειται στο γεγονός ότι η επιτάχυνση $p.acc$ δεν είναι σταθερή. Καθώς κινείται ο πλανήτης γύρω από τον ήλιο, ασκείται δύναμη, άρα επιτάχυνση, στον πλανήτη. Ως γνωστόν από τη Φυσική, η επιτάχυνση αυτή είναι στη διεύθυνση της ευθείας που ενώνει πλανήτη και ήλιο. Άρα σε κάθε βήμα θα μεταβάλλεται η

φορά της. Όμως και το μέτρο της θα μεταβάλλεται σε κάθε βήμα καθώς εξαρτάται από την απόσταση ήλιου-πλανήτη, απόσταση που συνεχώς θα επαναπροσδιορίζεται.

Σε κάθε βήμα πρέπει λοιπόν αρχικά να υπολογιστεί η επιτάχυνση και θα δοθούν εδώ οι απαραίτητες διευκρινήσεις. Μετά τον υπολογισμό της επιτάχυνσης, ακολουθείται η ίδια διαδικασία υπολογισμού νέας ταχύτητας και νέας θέσης όπως στο προηγούμενο παράδειγμα. Τώρα, δοθέντων των θέσεων πλανήτη $p.pos$ και ήλιου $h.pos$, πώς υπολογίζεται η επιτάχυνση $p.acc$;

Για να απαντηθεί το ερώτημα πρέπει να επιστρατευτούν μαθηματικές γνώσεις περί διανυσμάτων καθώς και γνώσεις Φυσικής περί βαρυτικής έλξης. Η απόσταση ήλιου-γης περιγράφεται διανυσματικά ως $h.pos - p.pos$. Για την κατανόηση αυτού, ας δούμε το παρακάτω σχήμα όπου φαίνεται ότι τα δύο συνεχόμενα διανύσματα $-p.pos$ και $h.pos$ συναθροίζονται τελικά σε ένα διάνυσμα ph που συνδέει πλανήτη και ήλιο.



Θα μας χρειαστεί το μέτρο αυτού του διανύσματος απόστασης. Γενικά, το μέτρο ενός διανύσματος υπολογίζεται σύμφωνα με το πυθαγόρειο θεώρημα ως ρίζα του αθροίσματος των τετραγώνων των συντεταγμένων του. Στην περίπτωσή μας το μέτρο του ph υπολογίζεται σε ρίζα του $ph[0]^2 + ph[1]^2$.

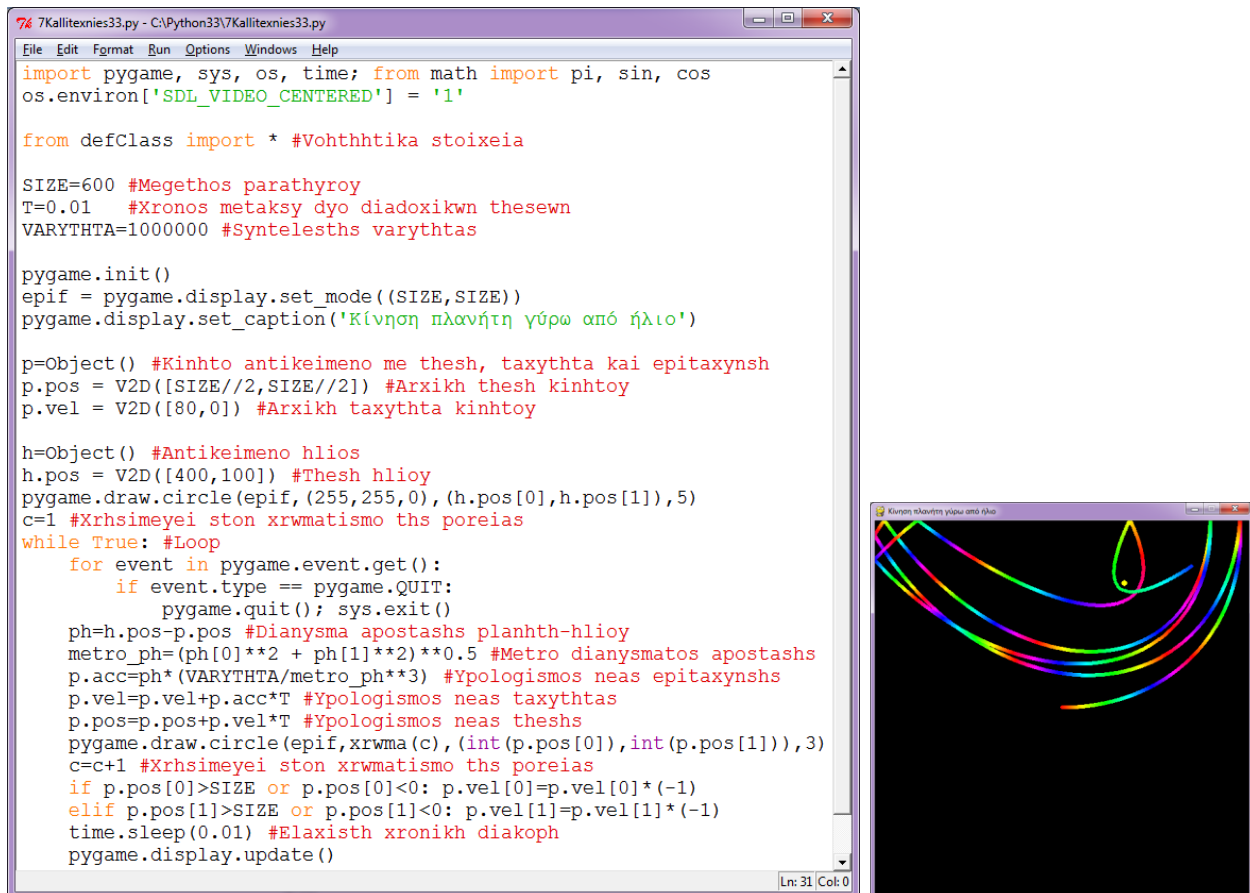
Επίσης θα μας χρειαστεί το μοναδιαίο διάνυσμα του διανύσματος απόστασης, διότι πρόκειται ταυτόχρονα για το μοναδιαίο διάνυσμα της επιτάχυνσης και χαρακτηρίζει ουσιαστικά τη φορά της επιτάχυνσης $p.acc$. Γενικά, το μοναδιαίο διάνυσμα ενός οποιουδήποτε διανύσματος υπολογίζεται αν διαιρέσουμε το διάνυσμα με το μέτρο του. Στην περίπτωσή μας το μοναδιαίο αυτό διάνυσμα είναι $ph/μέτρο(ph)$. Γράφεται ισοδύναμα και ως $ph*(1/μέτρο(ph))$.

Επίσης θα μας χρειαστεί το μέτρο της επιτάχυνσης. Είναι γνωστό από τους νόμους βαρύτητας της Φυσικής ότι η δύναμη που ασκείται σε ένα σώμα από ένα άλλο λόγω βαρύτητας δίνεται από τύπο της μορφής $F = d * m1 * m2 / s^2$ (όπου d σταθερός συντελεστής, $m1$ και $m2$ οι μάζες των σωμάτων και s η απόσταση). Καθώς η επιτάχυνση είναι ανάλογη της δύναμης ($F = m1 * a$), αυτό γράφεται ισοδύναμα $a = d * m2 / s^2$. Γενικά, λοιπόν, η επιτάχυνση που προκαλείται έχει μέτρο αντιστρόφως ανάλογο με την απόσταση s , κάτι που περιεκτικά μπορούμε να διατυπώσουμε ως $a = VARYTHTA / s^2$ (ή αλλιώς $a = VARYTHTA / μέτρο(ph)^2$ με $VARYTHTA$ να είναι ένας κατάλληλος σταθερός συντελεστής, ο οποίος εκφράζει το μέγεθος της αντίστροφης αναλογίας).

Αν τώρα πολλαπλασιάσουμε το μοναδιαίο διάνυσμα $ph/μέτρο(ph)$ με το μέτρο της επιτάχυνσης $VARYTHTA / μέτρο(ph)^2$ προκύπτει το τελικό πλήρες διάνυσμα $p1.acc$.

Συνολικά: $p1_{acc} = p1_{\text{μέτρο}}(p1) * \text{VARYTHTA}/\text{μέτρο}(p1)^2 = p1 * (\text{VARYTHTA}/\text{μέτρο}(p1))^3$

Όλα τα παραπάνω βρίσκουν εφαρμογή στο παρακάτω πρόγραμμα 7Kallitexnies33.py το οποίο γράφουμε και εκτελούμε.



```
7Kallitexnies33.py - C:\Python33\7Kallitexnies33.py
File Edit Format Run Options Windows Help

import pygame, sys, os, time; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'

from defClass import * #Voththtika stoixeia

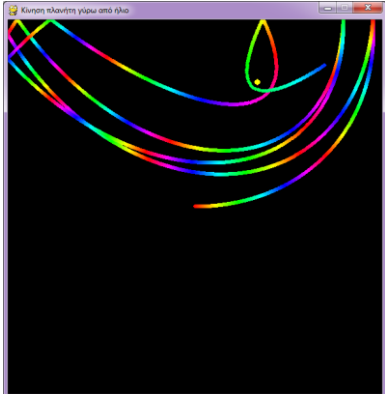
SIZE=600 #Megethos parathyroy
T=0.01 #Xronos metaksy dyo diadoxikwn thesewn
VARYTHTA=1000000 #Syntelesths varythtas

pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Κίνηση πλανήτη γύρω από ήλιο')

p=Object() #Kinhto antikeimeno me thesh, taxythta kai epitaxyntsh
p.pos = V2D([SIZE//2,SIZE//2]) #Arxikh thesh kinhtoy
p.vel = V2D([80,0]) #Arxikh taxythta kinhtoy

h=Object() #Antikeimeno hlios
h.pos = V2D([400,100]) #Thesh hlioy
pygame.draw.circle(epif, (255,255,0), (h.pos[0],h.pos[1]), 5)
c=1 #Xrhisimeyei ston xrwmatismo ths poreias
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()

    ph=h.pos-p.pos #Dianysma apostashs planhth-hlioy
    metro_ph=(ph[0]**2 + ph[1]**2)**0.5 #Metro dianysmatos apostashs
    p.acc=ph*(VARYTHTA/metro_ph**3) #Ypologismos neas epitaxyntshs
    p.vel=p.vel+p.acc*T #Ypologismos neas taxythtas
    p.pos=p.pos+p.vel*T #Ypologismos neas theshs
    pygame.draw.circle(epif, xrwma(c), (int(p.pos[0]),int(p.pos[1])), 3)
    c=c+1 #Xrhisimeyei ston xrwmatismo ths poreias
    if p.pos[0]>SIZE or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
    elif p.pos[1]>SIZE or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
    time.sleep(0.01) #Elaxisth xronikh diakoph
    pygame.display.update()
```



Καταρχήν η εντολή `from defClass import *` φροντίζει ώστε τα περιεχόμενα του αρχείου `defClass.py` να είναι διαθέσιμα (υποπρογράμματα χρωματισμών και αντικείμενα), και μιλήσαμε γι' αυτό στην αρχή του κεφαλαίου. Η νέα εντολή `VARYTHTA=1000000` ορίζει τον σταθερό συντελεστή που χρειαζόμαστε για τους υπολογισμούς της επιτάχυνσης παρακάτω. Μπορεί κανείς να δοκιμάσει και άλλες τιμές, όμως θα πρέπει να προσέξει ώστε να παίρνει αποδεκτά αποτελέσματα εντός του παραθύρου και με αποδεκτές ταχύτητες. Παρακάτω, η εντολή `p.pos=V2D([SIZE//2,SIZE//2])` ουσιαστικά τοποθετεί αρχικά το κινητό αντικείμενο (πλανήτη) στο κέντρο του παραθύρου, ενώ η `p.vel=V2D([80,0])` του δίνει μια αρχική ταχύτητα με φορά προς τα δεξιά. Στη συνέχεια η εντολή `h.pos=V2D([400,100])` τοποθετεί έναν ήλιο στο σημείο με συντεταγμένες (400,100) και η επόμενη εντολή `pygame.draw.circle` σχεδιάζει σε αυτό το σημείο ένα μικρό κίτρινο κύκλο. Η εντολή `c=1` καθώς και οι παρακάτω εντός του `Loop` εντολές `pygame.draw.circle(epif,xrwma(c))...` και `c=c+1` χρησιμεύουν στο πέρασμα διαφορετικών χρωματισμών στην πορεία του πλανήτη και η σχετική μέθοδος έχει εξηγηθεί σε προηγούμενο πρόγραμμα. Εντός του `Loop`, τώρα, οι εντολές που εκτελούνται αέναα. Αρχικά το σύνθημα και απαραίτητο μπλοκ εντολών της `for`.

Κατόπιν, ακολουθούν οι εντολές που υλοποιούν αυτά που εξηγήθηκαν στην αρχή του κεφαλαίου. Η εντολή `ph=h.pos-p.pos` δημιουργεί το διάνυσμα της απόστασης πλανήτη με ήλιο. Η εντολή `metro_ph=(ph[0]**2+ph[1]**2)**0.5` υπολογίζει το μέτρο αυτής της απόστασης. Σημειώνεται ότι οι δύο αστερίσκοι υποδηλώνουν ύψωση σε δύναμη και

υπενθυμίζεται από τα Μαθηματικά ότι ύψωση αριθμού σε 0.5 ισοδυναμεί με τη ρίζα αυτού του αριθμού. Η εντολή `p.acc=ph*(VARYTHTA/metro_ph**3)` είναι ουσιαστικά το τελευταίο στάδιο προς τον υπολογισμό της ζητούμενης επιτάχυνσης. Και αυτή εξηγήθηκε αναλυτικά στην αρχή. Όπως προειπώθηκε, η επιτάχυνση πρέπει να υπολογίζεται σε κάθε βήμα ξανά, καθώς οι αλλαγές στις θέσεις του πλανήτη και του ήλιου προκαλούν και συνεχή αλλαγή της. Δοθείσης της νέας `p.acc`, μπορεί στη συνέχεια η εντολή `p.vel=p.vel+p.acc*T` να υπολογίσει τη νέα ταχύτητα και έχουν δοθεί οι εξηγήσεις στο κεφάλαιο 32. Τώρα, δοθείσης της νέας `p.vel`, μπορεί στη συνέχεια η επόμενη εντολή `p.pos=p.pos+p.vel*T` να υπολογίσει τη νέα θέση του πλανήτη και έχουν δοθεί οι εξηγήσεις στο κεφάλαιο 31. Στη συνέχεια στη θέση αυτή με την `pygame.draw.circle` σχεδιάζεται ένα έγχρωμο στίγμα. Οι εντολές `if` και `elif` ελέγχουν, όπως είπαμε, να μην ξεφύγει ο πλανήτης εκτός παραθύρου, η εντολή `time.sleep(0.01)` προσδίδει μια μικρή χρονοκαθυστερήση και ακολουθεί η συνήθης και αναγκαία τελευταία εντολή του `Loop`, `pygame.display.update()`.

ΚΕΦΑΛΑΙΟ 34, Εφέ κομήτη

Πριν προχωρήσουμε στο βασικό ζήτημα του κεφαλαίου αυτού, ας κάνουμε κάποιες προσθήκες στο βοηθητικό αρχείο `defClass.py`. Στους χρωματισμούς προσθέτουμε το παρακάτω τμήμα.

```
def xrwmaRMWYR(a):
    '''Dexetai enan akeraio arithmo kai dhmiourgei thn antistoixh apoxrwsh
    me poreia apoxrwsewn kokkino, magenta, leyko, kitrino, kokkino ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 64 #To 64 xrhsimopoeitai dioti moirazei sto peripoy
    ypD = a % 64 #thn perioxh 0 ews 255 se 4 isa merh.
    if akM == 0: c = (255, 0, ypD*255/64)
    elif akM == 1: c = (255, ypD*255/64, 255)
    elif akM == 2: c = (255, 255, 255-ypD*255/64)
    elif akM == 3: c = (255, 255-ypD*255/64, 0)
    return c

def xrwmaRMWYWR(a):
    '''Dexetai enan akeraio arithmo kai dhmiourgei thn antistoixh apoxrwsh
    me poreia apoxrwsewn kokkino, magenta, leyko, kitrino, leyko, kokkino ktl.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 64 #To 64 xrhsimopoeitai dioti moirazei sto peripoy
    ypD = a % 64 #thn perioxh 0 ews 255 se 4 isa merh.
    if akM == 0: c = (255, 0, ypD*255/64)
    elif akM == 1: c = (255, ypD*255/64, 255)
    elif akM == 2: c = (255, 255, 255-ypD*255/64)
    elif akM == 3: c = (255, 255-ypD*255/64, 255-ypD*255/64)
    return c
```

Παρατηρεί κανείς ότι σε αυτούς τους χρωματισμούς υπάρχουν τέσσερις περιοχές μετάβασης. Γι' αυτό το λόγο και οι διαιρέσεις γίνονται γενικά με 64. Στο δεύτερο χρωματισμό `xrwmaRMWYWR()` ο προσεκτικός παρατηρητής ίσως εντοπίσει μια μικρή παραβίαση των κανόνων. Η τελευταία γραμμή θα έπρεπε να είναι με 0 στην μπλε συνιστώσα:

```
elif akM == 3: c = (255, 255-ypD*255/64, 0).
```

Όμως, επειδή στα όρια μετάβασης πρόκειται για ανοιχτόχρωμους χρωματισμούς (λευκό και κίτρινο) μπορεί να δημιουργηθεί αυτή η εξαίρεση στον κανόνα. Έτσι ουσιαστικά μετά το κίτρινο επανέρχεται το λευκό και μετά οι χρωματισμοί οδηγούνται προς το κόκκινο.

Στο μέρος `Class` των διανυσματικών αντικειμένων προσθέτουμε τα τμήματα `def metro` και `def monadiaio` όπως φαίνεται παρακάτω:

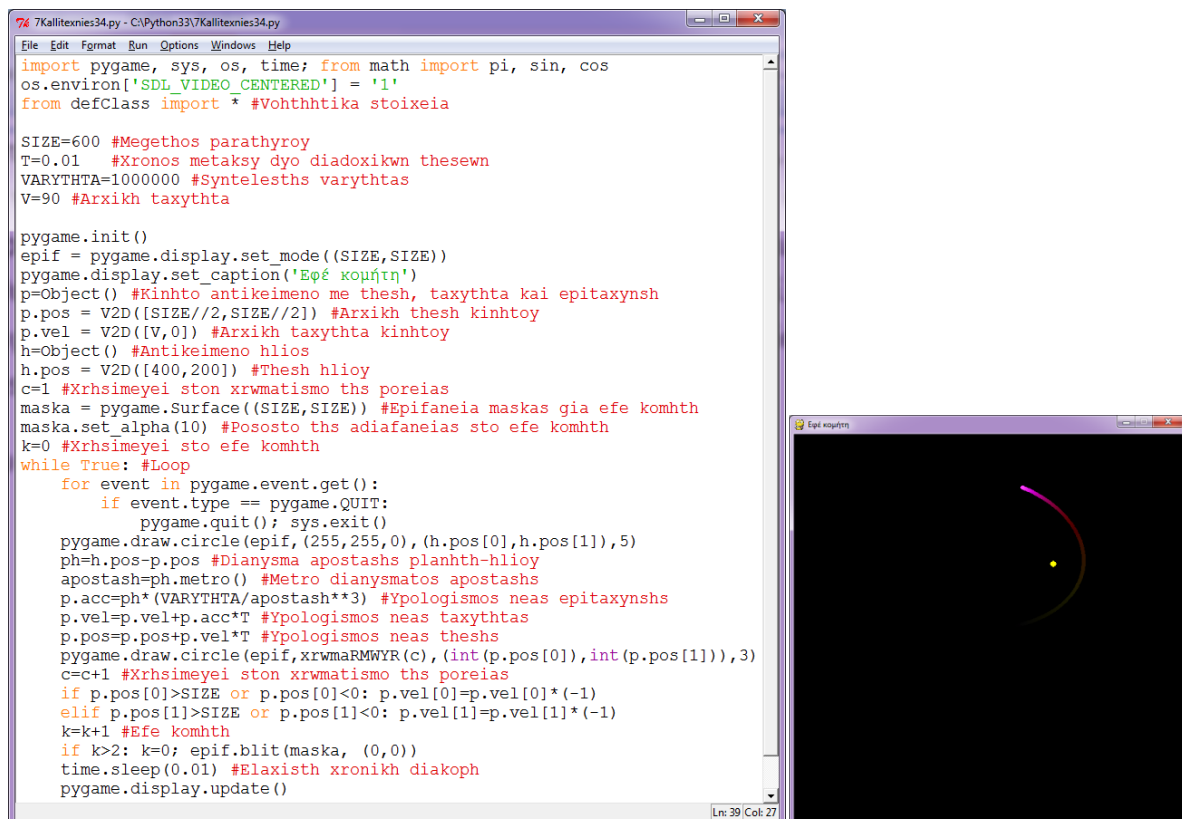
```

class V2D():
    '''Antikeimena dianysmata dyo diastasewn'''
    def __init__(self, data):
        self.data = data
    def __repr__(self):
        return repr(self.data)
    def __setitem__(self, key, item):
        self.data[key] = item
    def __getitem__(self, key):
        return self.data[key]
    def __add__(self, other):
        return V2D([self.data[0] + other.data[0], self.data[1] + other.data[1]])
    def __sub__(self, other):
        return V2D([self.data[0] - other.data[0], self.data[1] - other.data[1]])
    def __mul__(self, other):
        return V2D([self.data[0] * other, self.data[1] * other])
    def metro(self):
        return (self.data[0]**2 + self.data[1]**2)**0.5
    def monadiaio(self):
        m=(self.data[0]**2 + self.data[1]**2)**0.5
        if m!=0: return V2D([self.data[0]/m, self.data[1]/m])
        else: return V2D([0,0])

```

Αυτό μας δίνει τη δυνατότητα να υπολογίζονται σχεδόν αυτόματα το μέτρο και το μοναδιαίο διάνυσμα οποιουδήποτε διανύσματος όταν αυτά χρειάζονται. Έτσι, παρακάτω στον κώδικα θα δούμε να δημιουργείται πλέον το μέτρο του διανύσματος της απόστασης ρη με την απλή αναφορά `ph.metro()`. Η μορφή αυτή είναι χαρακτηριστική για μεθόδους αντικειμένων. Ως πρόθεμα μπαίνει η ονομασία του αντικειμένου, ακολουθεί τελεία και μετά ακολουθεί η επιθυμητή διεργασία σε μορφή υποπρογράμματος-μεθόδου που έχει δηλωθεί στα σχετικά αντικείμενα Class.

Γράφουμε το παρακάτω κυρίως πρόγραμμα 7Kallitexnies34.py και εκτελούμε.



```

7Kallitexnies34.py - C:\Python33\7Kallitexnies34.py
File Edit Format Run Options Windows Help

import pygame, sys, os, time; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'
from defClass import * #Vothhtika stoixeia

SIZE=600 #Megethos parathyroy
T=0.01 #Xronos metaksy dyo diadoxikwn thesewn
VARYTHTA=1000000 #Syntelesths varythtas
V=90 #Arxikh taxyhtta

pygame.init()
epif = pygame.display.set_mode((SIZE,SIZE))
pygame.display.set_caption('Εφέ κομήτη')
p=Object() #Kinhto antikeimeno me thesh, taxyhtta kai epitaxynsh
p.pos = V2D([SIZE//2,SIZE//2]) #Arxikh thesh kinhtoy
p.vel = V2D([V,0]) #Arxikh taxyhtta kinhtoy
h=Object() #Antikeimeno hlios
h.pos = V2D([400,200]) #Thesh hlioy
c=1 #Xrhsimeyei ston xrwmatismo ths poreias
maska = pygame.Surface((SIZE,SIZE)) #Epifaneia maskas gia efe komhth
maska.set_alpha(10) #Pososto ths adiafaneias sto efe komhth
k=0 #Xrhsimeyei sto efe komhth
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    pygame.draw.circle(epif, (255,255,0), (h.pos[0],h.pos[1]),5)
    ph=h.pos-p.pos #Dianysma apostashs planhth-hlioy
    apostash=ph.metro() #Metro dianysmatos apostashs
    p.acc=ph*(VARYTHTA/apostash**3) #Ypologismos neas epitaxynshs
    p.vel=p.vel+p.acc*T #Ypologismos neas taxyhttas
    p.pos=p.pos+p.vel*T #Ypologismos neas theshs
    pygame.draw.circle(epif, xrwmaRWYR(c), (int(p.pos[0]),int(p.pos[1])),3)
    c=c+1 #Xrhsimeyei ston xrwmatismo ths poreias
    if p.pos[0]>SIZE or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
    elif p.pos[1]>SIZE or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
    k=k+1 #Efe komhth
    if k>2: k=0; epif.blit(maska, (0,0))
    time.sleep(0.01) #Elaxisth xronikh diakoph
    pygame.display.update()

```

Παρατηρούμε ότι το κινητό κάνει την κίνησή του αφήνοντας πίσω του ένα είδος ουράς, γι' αυτό και ας ονομάσουμε αυτό το τέχνασμα «εφέ κομήτη». Πέντε νέες γραμμές κώδικα είναι υπεύθυνες γι' αυτό. Η εντολή `maska=pygame.Surface((SIZE,SIZE))` δημιουργεί μια νέα μαύρη επιφάνεια, ίδιου εμβαδού με το βασικό παράθυρο `epif`. Η επόμενη εντολή

`maska.set_alpha(10)` καθορίζει ότι αυτή η νέα επιφάνεια θα έχει αδιαφάνεια σε κάποιο μικρό ποσοστό της ενώ κατά το υπόλοιπο θα είναι διαφανής. Τοποθετώντας μια τέτοια επιφάνεια επάνω σε μια άλλη, πετυχαίνουμε ένα είδος μάσκας, δηλαδή η από κάτω επιφάνεια σκοτεινιάζει ελαφρά. Επανειλημμένη τοποθέτηση της επιφάνειας μάσκας πάνω σε άλλη, έχει ως αποτέλεσμα να σκοτεινιάζουν σιγά σιγά εντελώς όλα τα σημεία στην κάτω επιφάνεια, εκτός από τα φρεσκοβαμμένα. Η εντολή `k=0` χρησιμεύει στο εφέ κομήτης, διότι χρειαζόμαστε αργότερα ένα μετρητή των επαναλήψεων, ώστε να ορίσουμε κάθε πότε επιθυμούμε εφαρμογή της μάσκας. Αυτός ο μετρητής αυξάνεται σε κάθε επανάληψη όπως φαίνεται παρακάτω στην εντολή `k=k+1`. Στην περίπτωση μας, όταν ο μετρητής γίνεται 3 (άρα κάθε τρεις επαναλήψεις), εκτός από το ότι ο μετρητής ξαναμηδενίζεται με την `k=0`, εφαρμόζεται και η μάσκα με την εντολή `epif.blit(maska, (0,0))`. Η `blit` είναι μια μέθοδος που εφαρμόζεται σε αντικείμενα τύπου επιφάνειας και έχει ως αποτέλεσμα τη συγχώνευση μιας επιφάνειας πάνω σε μια άλλη. Στη περίπτωση μας η επιφάνεια `maska` επικάθεται και συγχωνεύεται πάνω στην επιφάνεια `epif` στο σημείο (0,0).

Παρατηρεί ίσως κανείς μια μικρή διαφορά, ότι η εντολή σχεδίασης του ήλιου `pygame.draw.circle(epif, (255,255,0), (h.pos[0],h.pos[1]), 5)`, μπήκε μέσα στο Loop. Αυτό γίνεται ώστε ο ήλιος να μην επηρεάζεται από την εφαρμογή της μάσκας και να είναι πάντα ορατός. Αν παρέμενε εκτός Loop, αργά αργά θα σκοτεινιάζε και θα εξαφανιζόταν. Επίσης έγινε μια αλλαγή στη θέση του ήλιου. Και επίσης η εντολή `v=90` στη θέση των σταθερών, καθορίζει το μέτρο της οριζόντιας αρχικής ταχύτητας. Για να ισχύει αυτό, πρέπει παρακάτω να συνοδεύεται από την εντολή `p.vel=v2D([v,0])`, κάτι που σημαίνει ότι η συντεταγμένη της ταχύτητας στον οριζόντιο άξονα *x* καθορίζεται ακριβώς από την τιμή που έχει η σταθερά *V*. Στον άξονα *y* δεν έχουμε αρχικά συντεταγμένη ταχύτητας.

ΚΕΦΑΛΑΙΟ 35, Πράσινος κομήτης

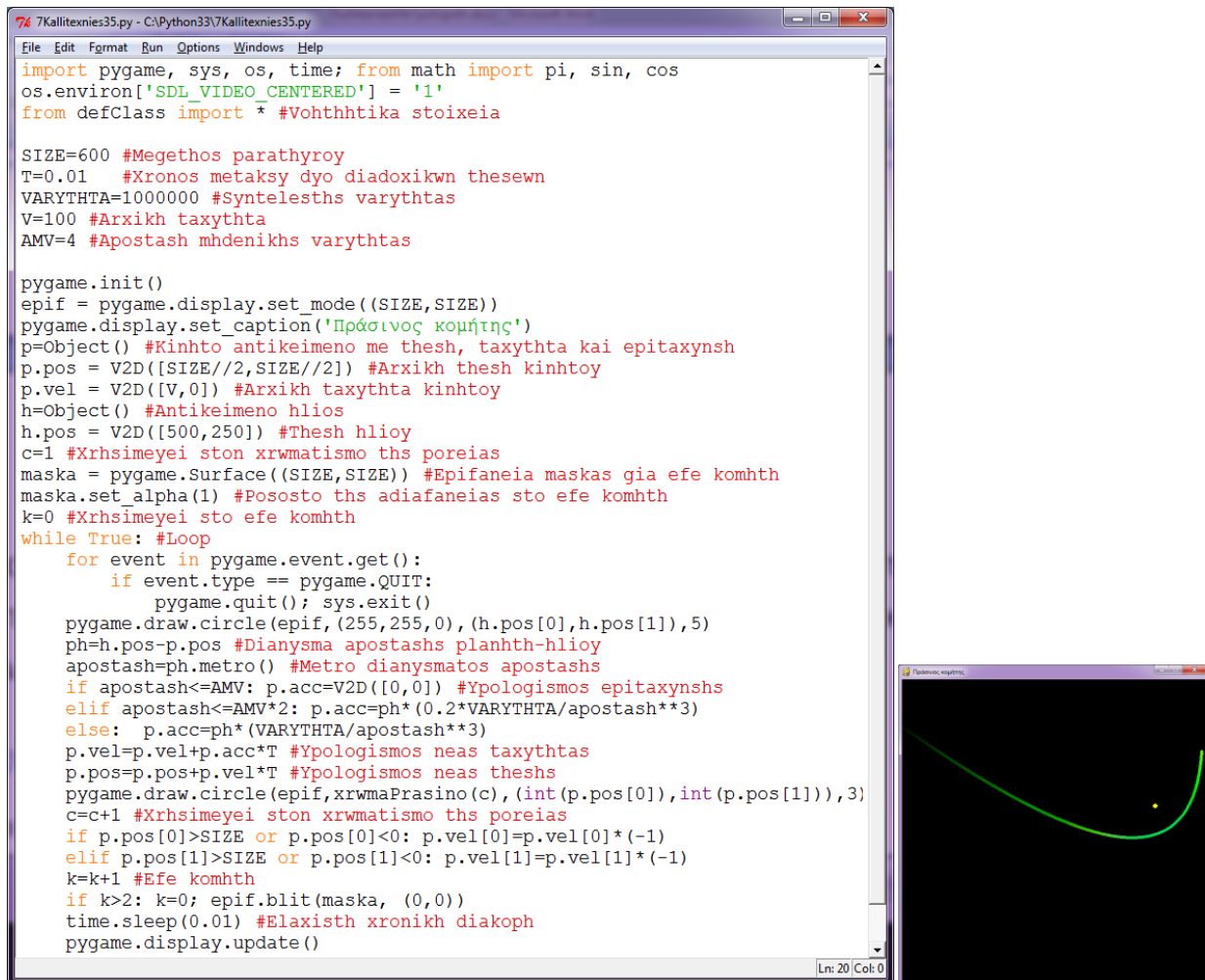
Ας κάνουμε μια προσθήκη στο βοηθητικό αρχείο `defClass.py`. Στους χρωματισμούς προσθέτουμε το παρακάτω τμήμα.

```
def xrwmaPrasino(a):
    '''Dexetai enan arithmo kai dhmioygei thn antistoixh apoxrwsh
    me apoxrwseis konta sto prasino.'''
    a = a % 255 #Diasfalizetai oti to a einai ews 255.
    akM = a // 64 #To 64 xrhsimopoietai dioti moirazei sto peripoy
    ypD = a % 64 #thn perioxh 0 ews 255 se 4 isa merh.
    if akM == 0: c = (ypD*127/64, 255, 0)
    elif akM == 1: c = (127, 255, ypD*127/64)
    elif akM == 2: c = (127-ypD*127/64, 255, 127)
    elif akM == 3: c = (0, 255, 127-ypD*127/64)
    return c
```

Παρατηρούμε ότι και στις τέσσερις περιοχές, παραμένει το 255 της πράσινης συνιστώσας. Στις άλλες δύο συνιστώσες, ίσως παρατηρεί κανείς ότι η μέγιστη τιμή που επιτυγχάνεται είναι 127. Αυτό έχει ως αποτέλεσμα να κυριαρχεί πάντα το πράσινο και να σχεδιάζονται αποχρώσεις λίγο πολύ κοντά σε αυτό.

Ίσως δεν είναι πλέον δύσκολο να δημιουργήσει κανείς και να προσθέσει στο αρχείο `defClass.py` και τα υποπρογράμματα `xrwmaKokkino` και `xrwmaMple` που θα διέπονται από παρόμοια λογική.

Γράφουμε τον παρακάτω κώδικα `7Kallitexnies35.py` και τον εκτελούμε.



Εκτός από την εντολή πράσινου χρωματισμού, που δεν χρειάζεται επεξηγήσεις έχει γίνει μια σημαντική τροποποίηση του κώδικα. Συγκεκριμένα στο σημείο που υπολογίζεται κάθε φορά η επιτάχυνση `p.acc`.

```
apostash=ph.metro() #metro dianysmatos apostashs
if apostash<=AMV: p.acc=V2D([0,0]) #ypologismos epitaxyynshs
elif apostash<=AMV*2: p.acc=ph*(0.2*VARYTHTA/apostash**3)
else: p.acc=ph*(VARYTHTA/apostash**3)
```

Αν τύχει και η τροχιά του κομήτη πλησιάσει πολύ κοντά στον ήλιο, τότε παρουσιάζεται το φαινόμενο, η πάρα πολύ κοντινή απόσταση να δημιουργεί τεράστιες επιταχύνσεις και κατά συνέπεια τεράστιες ταχύτητες που πλέον χαλάνε την ομαλή πορεία. Αυτό ακριβώς προλαμβάνουν οι τρεις παραπάνω γραμμές κώδικα. Η μεταβλητή `AMV` (απόσταση μηδενικής βαρύτητας) καθορίζει ένα ελάχιστο όριο απόστασης ήλιου-κομήτη. Αν τα δύο σώματα πλησιάσουν πιο κοντά από το όριο αυτό, τότε επιστρέφεται μηδενική επιτάχυνση. Αν τα σώματα βρίσκονται σε απόσταση `AMV` έως `2*AMV` τότε επιστρέφεται επιτάχυνση στο ένα πέμπτο της κανονικής, γι' αυτό βλέπουμε το συντελεστή `0.2`. Αν τα σώματα απέχουν περισσότερο από `2*AMV` τότε η επιτάχυνση υπολογίζεται και επιστρέφεται κανονικά. Αν για παράδειγμα έχει τεθεί `AMV=4`, τότε αν τα σώματα πλησιάσουν σε απόσταση μικρότερη ίση των 4 pixel, τότε δεν επιστρέφεται καθόλου επιτάχυνση. Αν τα σώματα πλησιάσουν σε απόσταση 5 έως 8 pixel, τότε με την εντολή `p.acc=ph*(0.2*VARYTHTA/apostash**3)`, επιστρέφεται το ένα πέμπτο της κανονικής επιτάχυνσης. Αποφεύγονται έτσι οι ενοχλητικές εμφανίσεις ακραίων καταστάσεων με υπερβολικές ταχύτητες.

Κατά τα άλλα δεν έχουν γίνει σοβαρές αλλαγές πέρα από τροποποίηση στην αρχική ταχύτητα του κομήτη, τροποποίηση στη θέση του ήλιου, και αλλαγή στη μάσκα ώστε να είναι ακόμα πιο διαφανής (`maska.set_alpha(1)`).

ΚΕΦΑΛΑΙΟ 36, Πολλές τροχιές

Έφτασε η στιγμή να προγραμματίσουμε πολύ εντυπωσιακές δημιουργίες. Προσωπικά, έμεινα έκπληκτος από αυτά που εκτυλίχθηκαν στην οθόνη του υπολογιστή μου, όταν αποφάσισα να προγραμματίσω όχι μία αλλά περισσότερες έγχρωμες γραμμές που θα κινούνται συμμετρικά μεταξύ τους. Ο προγραμματισμός, δεν ήταν πολύ δύσκολος. Το καλλιτεχνικό αποτέλεσμα όμως, δεν θα μπορούσε ποτέ να είχε προβλεφτεί. Απλά το είχα υποψιαστεί αμυδρά.

Αρχικά απαιτείται να συμπληρώσουμε στο τμήμα `class V2D()` του αρχείου `defClass.py`, δύο ακόμα υποπρογράμματα (`kart2pol` και `pol2kart`), διότι χρειάζεται πολλές φορές να μετατρέπουμε τα διανύσματα από μορφή καρτεσιανών συντεταγμένων σε μορφή πολικών συντεταγμένων και αντίστροφα. Επίσης απαιτείται να προστεθεί η γραμμή `from math`

```
from math import pi, sin, cos, atan2
class V2D():
    '''Antikeimena dianysmata dyo diastasewn'''
    def __init__(self, data):
        self.data = data
    def __repr__(self):
        return repr(self.data)
    def __setitem__(self, key, item):
        self.data[key] = item
    def __getitem__(self, key):
        return self.data[key]
    def __add__(self, other):
        return V2D([self.data[0] + other.data[0], self.data[1] + other.data[1]])
    def __sub__(self, other):
        return V2D([self.data[0] - other.data[0], self.data[1] - other.data[1]])
    def __mul__(self, other):
        return V2D([self.data[0] * other, self.data[1] * other])
    def metro(self):
        return (self.data[0]**2 + self.data[1]**2)**0.5
    def monadiaio(self):
        m=(self.data[0]**2 + self.data[1]**2)**0.5
        if m!=0: return V2D([self.data[0]/m, self.data[1]/m])
        else: return V2D([0,0])
    def kart2pol(self): #Metatrepei kartesianes se polikes syntetagmenes
        m=(self.data[0]**2 + self.data[1]**2)**0.5
        g=atan2(self.data[1],self.data[0])
        return V2D([m,g])
    def pol2kart(self): #Metatrepei polikes se kartesianes syntetagmenes
        m=self.data[0]
        return V2D([m*cos(self.data[1]), m*sin(self.data[1])])
```

Είναι γνωστό στα Μαθηματικά ότι ένα διάνυσμα στο επίπεδο μπορεί να περιγραφεί επακριβώς είτε με τις γνωστές καρτεσιανές συντεταγμένες, είτε όμως και με τις λεγόμενες πολικές συντεταγμένες. Στις πολικές, η πρώτη συντεταγμένη αφορά το μέτρο του διανύσματος, ενώ η δεύτερη αφορά τη γωνία που σχηματίζεται από το διάνυσμα και τον άξονα x. Πολλές φορές είναι πιο βολικές οι πολικές συντεταγμένες, ειδικά όταν θέλουμε να αναφερθούμε σε διανύσματα που διατάσσονται κυκλικά, συμμετρικά, και αυτό συμβαίνει στην περίπτωση μας. Έτσι, τα παραπάνω δύο υποπρογράμματα, θα αναλαμβάνουν να διεκπεραιώνουν τις σχετικές μετατροπές.

Γράφουμε τον παρακάτω κώδικα `7Kallitexnies36.py` και τον εκτελούμε.

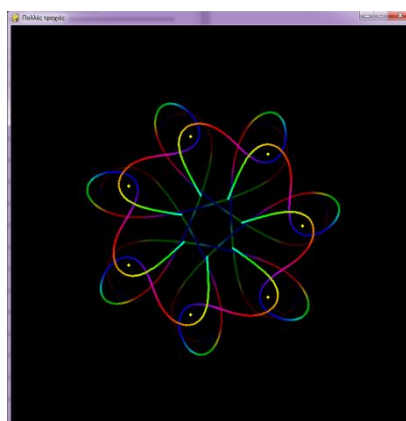
```

import pygame, sys, os, time; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'
from defClass import * #Vohthtika stoixeia

SZ=777 #Megethos parathyroy
A=177 #Aktina kykloy poy diatassontai oi hlioi
XD=0.01#Xronikh diakoph
T=0.01 #Xronos metaksy dyo diadoxikwn thesewn
VAR=777777 #Syntelesths varythtas
AMV=7 #Apostash mhdenikhs varythtas
N=7 #Plhthos hliwn, plhthos planhtwn
V=77 #Arxikh taxyhtta
M=7 #Moiros apoklishs troxiwn apo hlioy
PT=2 #Paxos troxiwn

pygame.init()
epif=pygame.display.set_mode((SZ,SZ))
pygame.display.set_caption('Πολλές τροχιές')
systhma=Object(); systhma.hlioi=[]; systhma.planhtes=[]; #Hliako systhma
for i in range(N):
    h=Object() #Dhmioyrgia hlioy
    h.pos=V2D([SZ//2+A*cos(i*2*pi/N),SZ//2+A*sin(i*2*pi/N)]) #Thesh hlioy
    systhma.hlioi.append(h) #O hlios prostithetai se lista
for i in range(N):
    p=Object() #Dhmioyrgia planhth
    p.pos=V2D([SZ//2,SZ//2]) #Arxikh thesh planhth
    p.vel=V2D([V,i*2*pi/N+2*pi*M/360]).pol2kart() #Arxikh taxyhtta planhth
    systhma.planhtes.append(p) #O planhts prostithetai se lista
c=1 #Xrhstimeyei ston xrwmatismo ths poreias
maska=pygame.Surface((SZ,SZ)) #Epifaneia maskas gia efe komhth
maska.set_alpha(1) #Pososto ths adiafaneias sto efe komhth
k=0 #Xrhstimeyei sto efe komhth
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    for h in systhma.hlioi:
        pygame.draw.circle(epif, (255,255,0), (int(h.pos[0]),int(h.pos[1])),3)
    for p in systhma.planhtes:
        p.acc=V2D([0,0])
        for h in systhma.hlioi:
            ph=h.pos-p.pos #Dianysma apostashs planhth-hlioy
            apostash=ph.metro() #Metro dianysmatos apostashs
            if apostash<=AMV: p.acc=p.acc+V2D([0,0]) #Ypologismos epitaxyynshs
            elif apostash<=AMV*2: p.acc=p.acc+ph*(0.2*VAR/apostash**3)
            else: p.acc=p.acc+ph*(VAR/apostash**3)
        p.vel=p.vel+p.acc*T #Ypologismos neas taxyhttas
        p.pos=p.pos+p.vel*T #Ypologismos neas theshs
        pygame.draw.circle(epif,xrwm(c), (int(p.pos[0]),int(p.pos[1])),PT//2)
        if p.pos[0]>SZ or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
        elif p.pos[1]>SZ or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
    c=c+1 #Xrhstimeyei ston xrwmatismo ths poreias
    k=k+1 #Efe komhth
    if k>2: k=0; epif.blit(maska, (0,0))
    time.sleep(XD) #Xronikh diakoph
    pygame.display.update()

```

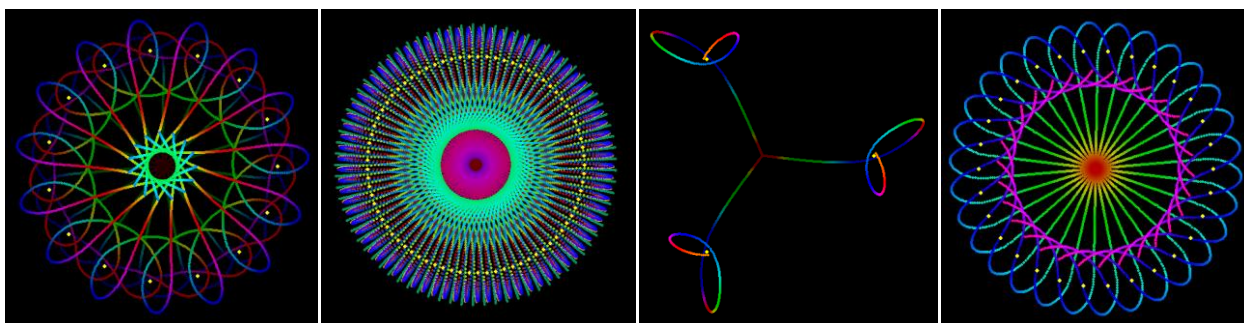


Ας δούμε το τμήμα με τις σταθερές:

```
SZ=777 #Megethos parathyroy  
A=177 #Aktina kykloy poy diatassontai oi hlioi  
XD=0.01#Xronikh diakoph  
T=0.01 #Xronos metaksy dyo diadoxikwn thesewn  
VAR=777777 #Syntelesths varythtas  
AMV=7 #Apostash mhdenikhs varythtas  
N=7 #Plhthos hliwn, plhthos planhtwn  
V=77 #Arxikh taxythta  
M=7 #Moiros apoklishs troxiwn apo hlioys  
PT=2 #Paxos troxiwn
```

Ουσιαστικά παίζει το ρόλο της παραμετροποίησης του προγράμματός μας. Για παράδειγμα εάν θέλουμε να τρέξουμε το ίδιο πρόγραμμα αλλά με 15 ήλιους, τότε απλά πρέπει να αλλάξουμε την τιμή της σταθεράς **N** που δηλώνει το πλήθος των ήλιων (κατά συνέπεια και πλήθος τροχιών). Εάν θέλουμε να αλλάξουμε την ακτίνα που διατάσσονται οι ήλιοι, τότε απλά πρέπει να αλλάξουμε την τιμή της σταθεράς **A**, και ούτω καθεξής.

Είναι εκπληκτικό το γεγονός ότι από τους άπειρους συνδυασμούς τιμών στις παραπάνω σταθερές, προκύπτουν ανάλογα και άπειρες δημιουργίες. Μπορώ να διαβεβαιώσω, ότι θα βρεθεί κανείς να χαζεύει με τις ώρες τις κινούμενες δημιουργίες στην οθόνη του.



Το **SZ** αντικατέστησε το προηγούμενο **SIZE** για λόγους συντόμευσης. Το **A** είναι η ακτίνα του κύκλου που διατάσσονται οι ήλιοι. Το **XD** χρησιμοποιείται στην προτελευταία εντολή **time.sleep** και αφορά τα δευτερόλεπτα της κάθε χρονικής διακοπής. Το **T** αφορά το συντελεστή χρόνου μεταξύ δύο διαδοχικών θέσεων, που θα χρησιμοποιηθεί στους τύπους της Φυσικής. Μεγαλύτερο **T** σημαίνει και αυξημένη ταχύτητα τροχιών και πιο αραιή σχεδίαση του στίγματος. Το **VAR** αντικατέστησε το προηγούμενο **VARYTHTA** και αφορά την τιμή που θα χρησιμοποιηθεί στους τύπους της Φυσικής, ως συντελεστής βαρύτητας και όπως μπορεί κανείς να διαπιστώσει, πρέπει να χρησιμοποιείται με σχετικά μεγάλες τιμές, π.χ. κοντά στο ένα εκατομμύριο. Το **AMV** είναι η απόσταση μηδενικής βαρύτητας, στην οποία πλέον δεν εφαρμόζεται δύναμη και επιτάχυνση έλξης των σωμάτων, διότι προκύπτουν ανεπιθύμητες καταστάσεις υπερβολικά μεγάλων ταχυτήτων. Για λόγους ομαλής μετάβασης από συνθήκες κανονικής επιτάχυνσης σε συνθήκες μηδενικής επιτάχυνσης, χρησιμοποιείται παρακάτω και η περιοχή **AMV** έως **2*AMV**, όπου εφαρμόζεται μειωμένη επιτάχυνση. Το **N** αφορά το πλήθος των ήλιων, κατά συνέπεια το πλήθος των πλανητών και των τροχιών τους. Το **V** αφορά το μέτρο της αρχικής ταχύτητας των πλανητών. Το **M** αφορά τη διαφορά σε μοίρες των αρχικών τροχιών από τα σημεία που βρίσκονται οι ήλιοι. Εάν δεν υπάρχει αυτή η διαφορά (**M=0**), τότε οι κινήσεις παρουσιάζονται μόνο ευθύγραμμες. Το **PT** αφορά το πάχος των τροχιών και βέβαια εμφανίζεται υπολογισμένο στο μισό ως τελευταίο όρισμα στην εντολή **pygame.draw.circle** που σχεδιάζει τα στίγματα των πλανητών, διότι η ακτίνα έχει το μισό του πάχους της τροχιάς.

Μετά τις σταθερές, ακολουθούν οι γνωστές μας εντολές `pygame.init()`, `epif=pygame.display.set_mode((SZ,SZ))` και `pygame.display.set_caption`.

Οι εντολές `systhma=Object()`, `systhma.hlioi=[]` και `systhma.planhtes=[]` δημιουργούν το βασικό πλαίσιο αντικειμένων του προγράμματος. Το αντικείμενο `systhma` ουσιαστικά είναι ένα ηλιακό σύστημα που περιλαμβάνει ήλιους και πλανήτες. Αρχικά δεν υπάρχουν ήλιοι και πλανήτες, γι' αυτό οι ορθογώνιες αγκύλες αρχικά δεν περιέχουν τίποτα. Οι ορθογώνιες αγκύλες μας εισάγουν σε ένα βασικό προγραμματιστικό αντικείμενο, αυτό της λίστας, το οποίο θα κατανοήσουμε με τον καιρό. Καθώς οι παραπάνω τρεις εντολές σχετίζονται μεταξύ τους, μπορούμε να τις γράψουμε στην ίδια γραμμή, διαχωρισμένες με ερωτηματικά.

Η εντολή `for` που ακολουθεί, δημιουργεί `N` ήλιους και με την εντολή `h.pos=V2D([SZ//2+A*cos(i*2*pi/N),SZ//2+A*sin(i*2*pi/N)])` σε καθέναν ορίζεται η θέση του, σύμφωνα και με όσα έχουν επεξηγηθεί στο κεφάλαιο 6 περί κυκλικής διάταξης σημείων. Επειδή η θέση αυτή πρέπει να μπορεί να διαχειρίζεται ως διάνυσμα, το ζεύγος των συντεταγμένων της θέσης πρέπει να αποκτήσει την ιδιότητα του διανύσματος και αυτό γίνεται με το πρόθεμα `V2D` ακολουθούμενο από τις συντεταγμένες σε παρένθεση. Η εντολή `systhma.hlioi.append(h)` προσθέτει κάθε δημιουργημένο ήλιο στη σχετική λίστα.

Η επόμενη εντολή `for` αφορά τη δημιουργία `N` πλανητών. Η εντολή `p.pos=V2D([SZ//2,SZ//2])` τοποθετεί όλους τους πλανήτες στο κέντρο του παραθύρου, ορίζοντας ως κατάλληλο διάνυσμα την ιδιότητα `p.pos` του κάθε πλανήτη. Η εντολή `p.vel=V2D([V,i*2*pi/N+2*pi*M/360]).pol2kart()` ορίζει τις αρχικές ταχύτητες των πλανητών, ορίζοντας ως κατάλληλο διάνυσμα την ιδιότητα `p.vel` κάθε πλανήτη. Οι συντεταγμένες ορίζονται αρχικά εύκολα ως πολικές και το υποπρόγραμμα `pol2kart()` τις μετατρέπει σε καρτεσιανές για λόγους συμβατότητας με το υπόλοιπο πρόγραμμα, όπου παντού χρησιμοποιούνται καρτεσιανές. Παρατηρεί κανείς ότι στη βασική γωνία $i*2\pi/N$ έχει προστεθεί και μια μικρή γωνία απόκλισης $2\pi*M/360$ (M μοιρών), ώστε οι αρχικές τροχιές να μην κατευθύνονται ακριβώς πάνω στους ήλιους. Μπορεί κανείς να δοκιμάσει να τρέξει το πρόγραμμα με $M=0$ ώστε να αντιληφθεί καλύτερα αυτό το ζήτημα. Τέλος, η εντολή `systhma.planhtes.append(p)` προσθέτει κάθε δημιουργημένο πλανήτη στη σχετική λίστα.

Ακολουθεί η εντολή `c=1` που χρησιμεύει στους χρωματισμούς και στη συνέχεια οι τρεις εντολές που χρησιμεύουν στο εφέ κομήτη.

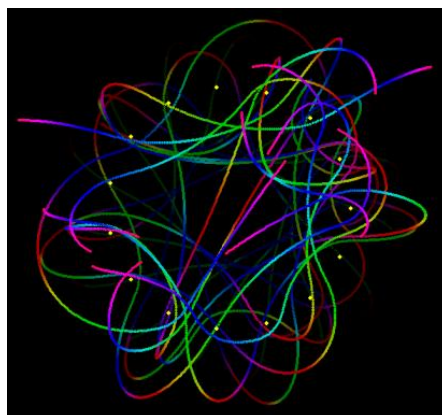
Μέσα στο `Loop`, η πρώτη `for` αφορά όπως έχει ειπωθεί τον εντοπισμό ενδεχόμενης ενέργειας κλεισίματος του προγράμματος.

Η δεύτερη `for` σχεδιάζει όλους τους ήλιους. Να έχουμε υπόψη ότι η επαναληπτική εντολή `for` δουλεύει και με το `range` (όπου καθορίζεται το πλήθος επαναλήψεων) και με λίστες (όπου οι επαναλήψεις θα αφορούν όλα τα αντικείμενα της λίστας).

Η τρίτη `for` είναι αυτή που κάνει την πολλή εργασία. Η εντολή `for p in systhma.planhtes` και οι εντολές της σε εσοχή, θα υπολογίζει και θα σχεδιάζει για κάθε πλανήτη την τρέχουσα θέση. Ουσιαστικά για κάθε πλανήτη θα υπολογίζεται αρχικά η νέα επιτάχυνση, κατόπιν η νέα ταχύτητα και τέλος η νέα θέση του, η οποία και θα σχεδιάζεται.

Όμως, ειδικά το πρώτο βήμα, ο υπολογισμός της επιτάχυνσης, πρέπει να λαμβάνει υπόψη τις επιταχύνσεις που προκαλούνται στον πλανήτη από κάθε ήλιο. Γι' αυτό το λόγο, κάθε φορά με την εντολή `p.acc=V2D([0,0])` η επιτάχυνση μηδενίζεται, ενώ στη συνέχεια η εντολή `for h in systma.hlioi` αυξάνει την επιτάχυνση κατάλληλα (`p.acc=p.acc+...`), ανάλογα της απόστασης του πλανήτη από τον κάθε ήλιο. Οι προσυζητήσεις θα είναι `N` στο πλήθος και τελικά θα προκύψει η τελική επιτάχυνση `p.acc`. Βάση αυτής, οι επόμενες εντολές `p.vel=p.vel+p.acc*T` και `p.pos=p.pos+p.vel*T` θα κάνουν την υπόλοιπη εργασία, όπως έχει εξηγηθεί και στα προηγούμενα κεφάλαια. Η εντολή `pygame.draw.circle` θα σχεδιάσει το στίγμα του κάθε πλανήτη και ας προσέξουμε ότι ως τελευταίο όρισμα μεγέθους ακτίνας έχει τοποθετηθεί η παράσταση `PT//2`. Και ακολουθεί η γνωστή `if` που ενδεχομένως επαναφέρει τις τροχιές που κατευθύνονται εκτός παραθύρου, αλλάζοντας τη φορά της ταχύτητας.

Όταν ολοκληρωθούν όλες οι τροχιές, η εντολή `c=c+1` θα φροντίσει ώστε ο επόμενος χρωματισμός να διαφέρει από τον επόμενο. Οι επόμενες εντολές `k=k+1` και `if k>2: k=0; epif.blit(maska,(0,0))`, αφορούν το γνωστό εφέ κομήτη. Η εντολή `time.sleep(XD)` είναι απλά η γενίκευση της εντολής του προηγούμενου προγράμματος `time.sleep(0.01)`, έτσι ώστε να μπορούμε από το τμήμα των σταθερών (συγκεκριμένα με αλλαγή στην τιμή της `XD`) να δοκιμάζουμε άλλες τιμές χρονικής διακοπής. Και τελικά η γνωστή `pygame.display.update()` ολοκληρώνει τις εντολές του Loop.



Όλα καλά κι ωραία, όμως έπειτα από κάποια δευτερόλεπτα παρατηρείται μια πλήρης κατάρρευση της κυκλικής συμμετρίας των τροχιών, όπως βλέπουμε παραπάνω π.χ. για `N=15`. Αυτό συμβαίνει διότι κατά τον υπολογισμό των τροχιών εφαρμόζεται στρογγυλοποίηση σε κλίμακα πίξελ. Στα πρώτα δευτερόλεπτα, οι κλασματικές αποκλίσεις στρογγυλοποιούνται ομαλά, δίχως προβλήματα, με συμμετρικό αποτέλεσμα. Σιγά σιγά όμως, επέρχονται αποκλίσεις του ενός ή και των δύο πίξελ, οι οποίες σε κάθε επόμενο βήμα προκαλούν κι άλλες αποκλίσεις. Ίσως αυτή η μετάβαση από την τάξη στο χάος να έχει τη γοητεία της. Τι μπορούμε να κάνουμε όμως αν δεν την επιθυμούμε;

ΚΕΦΑΛΑΙΟ 37, Συμμετρικές τροχιές

Μια λύση στο πρόβλημα που αναφέρθηκε στο τέλος του προηγούμενου κεφαλαίου είναι να υπολογίζεται μόνο μια τροχιά (και όχι όλες). Και στη συνέχεια να υπολογίζονται και να σχεδιάζονται οι υπόλοιπες με βάση την κυκλική συμμετρία.

Τροποποιούμε λοιπόν κατάλληλα το προηγούμενο πρόγραμμα, γράφουμε τον παρακάτω κώδικα `7Kallitexnies37.py` και τον εκτελούμε.


```

import pygame, sys, os, time; from math import pi, sin, cos
os.environ['SDL_VIDEO_CENTERED'] = '1'
from defClass import * #Vohthhtika stoixeia

SZ=777      #Megethos parathyroy
A=177       #Aktina kykloy poy diatassontai oi hlioi
XD=0.005    #Xronikh diakoph
T=0.01      #Xronos metaksy dyo diadoxikwn thesewn
VAR=777777 #Syntelesths varythtas
SKV=0.2     #Syntelesths kontinhs varythtas
AMV=7       #Apostash mndenikhs varythtas
N=5         #Plhthos hliwn, plhthos planhtwn
V=77       #Arxikh taxythta
M=7         #Moiros apoklishs troxiwn apo hlioy
PT=2        #Paxos troxiwn
DEK=5       #Diarkeia efe komhth
AX=1        #Arxikos xrwmatismos
TEX=0.1     #Taxythta enallaghs xrwmatos

pygame.init()
epif=pygame.display.set_mode((SZ,SZ))
pygame.display.set_caption('Kef37 SZ'+str(SZ)+' A'+str(A)+' XD'+str(XD)+' T'+str(T)+'
    ' VAR'+str(VAR)+' SKV'+str(SKV)+' AMV'+str(AMV)+' N'+str(N)+' V'+str(V)+' M'+str(M)+'
    ' PT'+str(PT)+' DEK'+str(DEK)+' AX'+str(AX)+' TEX'+str(TEX) )

s=Object(); s.hlioi=[]; s.planhtes=[]; #Hliako systhma
for i in range(N):
    h=Object() #Dhmiourgia hlioy
    h.pos=V2D([SZ//2+A*cos(i*2*pi/N),SZ//2+A*sin(i*2*pi/N)]) #Thesh hlioy
    s.hlioi.append(h) #O hlios prostithetai se lista

p=Object() #Dhmiourgia enos planhth
p.pos=V2D([SZ//2,SZ//2]) #Arxikh thesh planhth
p.vel=V2D([V,2*pi/N+2*pi*M/360]).pol2kart() #Arxikh taxythta planhth

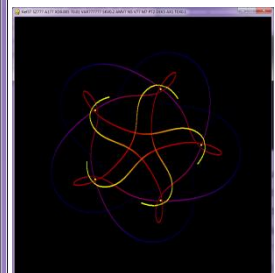
c=AX #Xrhsimeyei ston xrwmatismo ths poreias, AX einai o arxikos xrwmatismos
maska=pygame.Surface((SZ,SZ)) #Epifaneia maskas gia efe komhth
maska.set_alpha(1) #Pososto ths adiafaneias sto efe komhth
k=0 #Xrhsimeyei sto efe komhth
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
    for h in s.hlioi:
        pygame.draw.circle(epif, (255,255,0), (int(h.pos[0]),int(h.pos[1])),3)

    p.acc=V2D([0,0])
    for h in s.hlioi:
        ph=h.pos-p.pos #Dianysma apostashs planhth-hlioy
        apostash=ph.metro() #Metro dianysmatos apostashs
        if apostash<=AMV: p.acc=p.acc+V2D([0,0]) #Ypologismos epitaxynshs
        elif apostash<=AMV*2: p.acc=p.acc+ph*(SKV*VAR/apostash**3)
        else: p.acc=p.acc+ph*(VAR/apostash**3)
    p.vel=p.vel+p.acc*T #Ypologismos neas taxythtas
    p.pos=p.pos+p.vel*T #Ypologismos neas theshs
    if p.pos[0]>SZ or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
    elif p.pos[1]>SZ or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)

    for i in range(N): #Edw ypologizontai oi symmetrikes troxies
        ps=Object()
        ps.pos=(p.pos-V2D([SZ//2,SZ//2])).kart2pol() #p.pos se polikes apo kentro
        ps.pos=V2D([ps.pos[0],ps.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        ps.pos=V2D([ps.pos[0]+SZ//2,ps.pos[1]+SZ//2]) #ps.pos apo arxh twn aksonwn
        pygame.draw.circle(epif,xrwm(c), (int(ps.pos[0]),int(ps.pos[1])),PT//2)
        s.planhtes.append(ps)

    c=c+TEX #Xrhsimeyei ston xrwmatismo ths poreias, TEX einai taxythta enallaghs xrwmatos
    k=k+1 #Efe komhth
    if k>DEK: k=0; epif.blit(maska, (0,0)) #DEK einai h diarkeia efe komhth
    time.sleep(XD) #Xronikh diakoph
    pygame.display.update()

```



Σε σχέση με το προηγούμενο πρόγραμμα, παρατηρεί κανείς ότι πλέον αρχικά δημιουργείται μόνο ένας πλανήτης αντί για N . Βάση αυτού, υπολογίζονται μέσα στο Loop μία επιτάχυνση μία ταχύτητα και μία νέα θέση. Αυτή η νέα θέση, χρησιμοποιείται στο τμήμα **for i in range(N) :**, για να υπολογιστεί κάθε συμμετρική θέση της στο κύκλο. Για να βρεθεί κάθε συμμετρική θέση, είναι βολικό να μετατραπούν οι καρτεσιανές συντεταγμένες της παλιάς θέσης σε πολικές. Σημειώνεται ότι χρησιμοποιείται η παράσταση **p.pos-V2D([SZ//2,SZ//2])** διότι δεν θέλουμε διάνυσμα θέσης που να εκκινεί από την αρχή των αξόνων αλλά διάνυσμα που να εκκινεί από το κέντρο του παραθύρου. Από μια θέση σε πολικές συντεταγμένες, εύκολα υπολογίζεται μια συμμετρική της θέση σε κύκλο, διότι απαιτείται μόνο να προστεθεί στη δεύτερη συντεταγμένη η γωνία $2\pi/N$ κατάλληλες

φορές. Αυτό γίνεται στην εντολή `V2D([ps.pos[0],ps.pos[1]+i*2*pi/N])` και εν συνεχεία γίνεται πάλι μετατροπή σε καρτεσιανές συντεταγμένες. Τελικά, ο νεοδημιουργημένος πλανήτης με την ιδιότητα της νέας συμμετρικής θέσης προστίθεται στο ηλιακό μας σύστημα με την εντολή `s.planhtes.append(ps)` κάτι που δεν είναι και απαραίτητο. Τελικά σχεδιάζεται κάθε τροχιά.

Προστέθηκαν τέσσερις σταθερές στο τμήμα των παραμέτρων του προγράμματος. Η `SKV` σε συνδυασμό με την παρακάτω εντολή `elif apostash<=AMV*2:` `p.acc=p.acc+ph*(SKV*VAR/apostash**3)` (αντί για `0.2*VAR/apostash**3`) φροντίζει για την παραμετροποίηση της επιτάχυνσης σε κοντινές αποστάσεις. Η `DEK` σε συνδυασμό με την παρακάτω εντολή `if k>DEK:` (αντί για `if k>2:`), φροντίζει για την παραμετροποίηση στη διάρκεια του εφέ κομήτη. Η `AX` σε συνδυασμό με την παρακάτω εντολή `c=AX` (αντί για `c=1`) φροντίζει για την παραμετροποίηση στον αρχικό χρωματισμό. Η `TEX` σε συνδυασμό με την παρακάτω εντολή `c=c+TEX` (αντί για `c=c+1`) φροντίζει για την παραμετροποίηση στην ταχύτητα εναλλαγής των χρωματισμών.

Αξίζει να σημειωθεί ότι με την εντολή `pygame.display.set_caption` πλέον στον τίτλο του παραθύρου εξόδου εμφανίζονται όλες οι παράμετροι του προγράμματος και οι τιμές τους. Έτσι, μπορεί κανείς με `Alt+PrtScr` να αντιγράψει π.χ. το παράθυρο εξόδου με τις δημιουργίες του προς στο Word ή τη Ζωγραφική καταγράφοντας παράλληλα με ακρίβεια την παραμετροποίηση με την οποία επετεύχθη η δημιουργία του. Όμως υπάρχει κι άλλος τρόπος καταγραφής των παραμέτρων. Στο σημείο του Loop με τα συμβάντα πληκτρολογίου και ποντικιού, μέχρι στιγμής διαχειριζόμαστε μόνο συμβάν κλεισίματος. Θα προσθέσουμε λίγες γραμμές κώδικα, όπως φαίνεται παρακάτω, ώστε να διαχειριζόμαστε και πάτημα στο πλήκτρο `s` (για save) κάτι που θα δημιουργεί και θα αποθηκεύει ένα στιγμιότυπο σε αρχείο Ζωγραφικής bitmap με ονομασία όλη την παραμετροποίηση!

```
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit(); sys.exit()
    elif event.type == pygame.KEYDOWN and event.key == pygame.K_s:
        namebmp='Kef37 SZ'+str(SZ)+' A'+str(A)+' XD'+str(XD)+' T'+str(T)+' VAR'+str(VAR)\
        +' SKV'+str(SKV)+' AMV'+str(AMV)+' N'+str(N)+' V'+str(V)+' M'+str(M)+' PT'+str(PT)\
        +' DEK'+str(DEK)+' AX'+str(AX)+' TEX'+str(TEX)+' sec'+str(int(c*XD/TEX))+'.bmp'
        pygame.image.save(epif,namebmp)
```

Συμβάν πατήματος πλήκτρου (`event.type==pygame.KEYDOWN`) και μάλιστα πλήκτρο `s` (`event.key==pygame.K_s`) δημιουργεί αρχικά ένα τεράστιο όνομα `namebmp` με την παραμετροποίηση. Κατόπιν, η εντολή `pygame.image.save(epif,namebmp)` δημιουργεί από το παράθυρο `epif`, ένα αρχείο Ζωγραφικής τύπου bmp από το στιγμιότυπο τη στιγμή που πατήθηκε το πλήκτρο, δίνοντας σε αυτό το αρχείο το όνομα `namebmp` που περιέχει την παραμετροποίηση. Η παράσταση `' sec'+str(int(c*XD/TEX))` που προστίθεται στην παραμετροποίηση, είναι ένα χρονικό μέγεθος (σε δευτερόλεπτα), το οποίο δείχνει πόσα περίπου δευτερόλεπτα έχουν περάσει από την αρχή έως τη στιγμή που πατιέται το πλήκτρο `s`. Γι αυτό χρησιμεύει η τιμή της μεταβλητής `c` που συνεχώς αυξάνεται, όμως πολλαπλασιάζεται με το `XD` (χρόνος διακοπής) και διαιρείται με το `TEX` (το στοιχειώδες χρωματικό βήμα). Στα πλεονεκτήματα αυτής της δεύτερης μεθόδου αποτύπωσης της παραμετροποίησης (σε σχέση με την πρώτη με `Alt+PrtScr`), συμπεριλαμβάνεται και το γεγονός ότι μπορούν να δημιουργηθούν αρχεία που δεν χωρούν στην οθόνη, π.χ. μεγέθους 2000pixelx2000pixel δηλώνοντας μεγάλο `SZ` και μεγάλη ακτίνα δράσης `A` των τροχιών.

Αν θέλουμε οι τροχιές να σχηματίζονται αντί από κυκλικά στίγματα, από μικρότατα ευθύγραμμα τμήματα (για λόγους οπτικά καλύτερων τροχιών), τότε δημιουργούμε με τις λίγες αλλαγές στα παρακάτω κόκκινα σημεία το πρόγραμμα 7Kallitexnies37b.py.

```

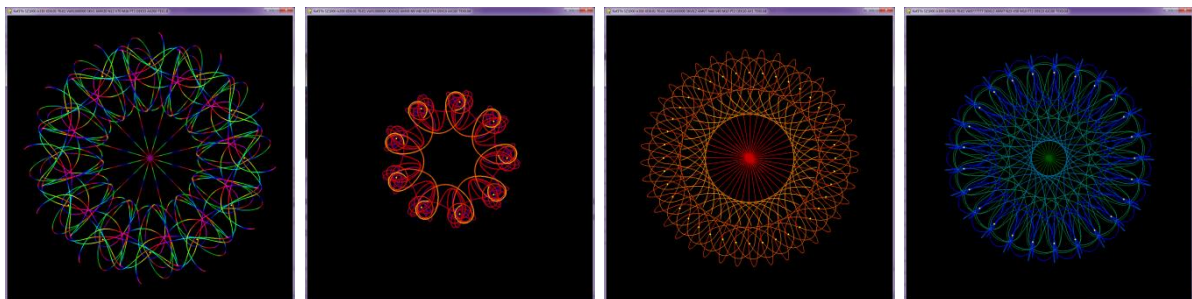
    elif apostash==AMV^2: p.acc=p.acc*ph*(SNV^VAR/apostash**3)
    else: p.acc=p.acc+ph*(VAR/apostash**3)
p.vel=p.vel+p.acc*T #Ypologismos neas taxyhtas
• p.posA=p.pos
p.pos=p.pos+p.vel*T #Ypologismos neas theshs
if p.pos[0]>SZ or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
elif p.pos[1]>SZ or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)

for i in range(N): #Edw ypologizontai oi symmetrikes troxies
ps=Object()
• ps.posA=(p.posA-V2D((SZ//2,SZ//2))).kart2pol() #ps.posA se polikes apo kentro
• ps.posA=V2D((ps.posA[0],ps.posA[1]+i*2*pi/N)).pol2kart() #Athroish i gwniwn
• ps.posA=V2D((ps.posA[0]+SZ//2,ps.posA[1]+SZ//2)) #ps.posA apo arxh tw aksonwn
ps.pos=(p.pos-V2D((SZ//2,SZ//2))).kart2pol() #ps.se polikes apo kentro
ps.pos=V2D((ps.pos[0],ps.pos[1]+i*2*pi/N)).pol2kart() #Athroish i gwniwn
ps.pos=V2D((ps.pos[0]+SZ//2,ps.pos[1]+SZ//2)) #ps.pos apo arxh tw aksonwn
• pygame.draw.line(epif,xrwna(c),(ps.posA[0],ps.posA[1]),(ps.pos[0],ps.pos[1]),PT)
s.planhtes.append(ps)

c=c+TEX #Xrhstimeyei ston xrwmatismo ths poreias, TEX einai taxyhta enallagis xrwmatos
k=k+1 #Efe komhth
if k>DEK: k=0; epif.blit(maska,(0,0)) #DEK einai h diarkeia efe komhth
time.sleep(XD) #Xronikh diakoph
pygame.display.update()

```

Αυτό που ουσιαστικά γίνεται είναι να κρατείται η προηγούμενη θέση στο `p.posA` πριν υπολογιστεί η επόμενη θέση `p.pos`. Βάση αυτών, υπολογίζονται στη συνέχεια οι συμμετρικές θέσεις τους. Και τελικά σχεδιάζεται μια γραμμή που ενώνει τη κάθε θέση (`ps.posA[0],ps.posA[1]`) με κάθε επόμενη της (`ps.pos[0],ps.pos[1]`).



Παραπάνω βλέπουμε μερικές δημιουργίες με το 7Kallitexnies37b.py.

Κάπου εδώ, προσωπικά αντιλήφθηκα ότι η δημιουργικότητα των προγραμμάτων είναι απεριόριστη! Αναρωτήθηκα π.χ. τι γίνεται άραγε αν προγραμματίσουμε γραμμές που δεν ενώνουν απλά προηγούμενη και επόμενη θέση, αλλά γραμμές που ενώνουν τη θέση `p.pos` με τη θέση που προκύπτει από το άθροισμα `p.pos+p.vel` (`p.posB`). Τροποποιούμε ελάχιστα το προηγούμενο πρόγραμμα 7Kallitexnies37b.py στα κόκκινα σημάδια, γράφοντας τον παρακάτω κώδικα 7Kallitexnies37c.py και τον δοκιμάζουμε με διάφορες παραμέτρους (για να μην υπολογίζονται θέσεις εκτός παραθύρου, συνιστάται μεγάλο AMV, μικρό V ή δοκιμές με άλλες αλλαγές παραμέτρων με αντίστοιχο αποτέλεσμα). Η θέση `p.posA` τώρα δεν θα χρησιμοποιηθεί.

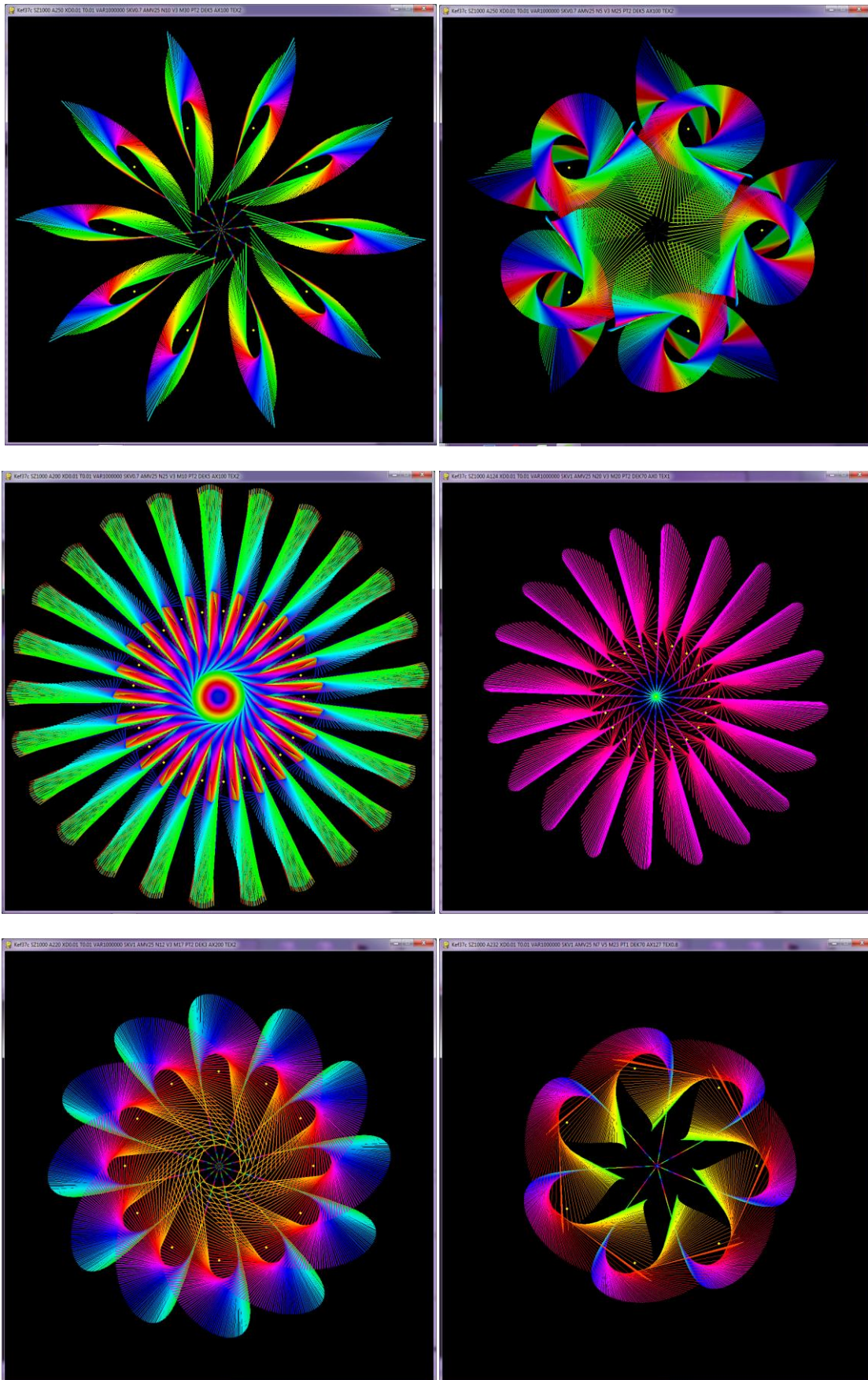
```

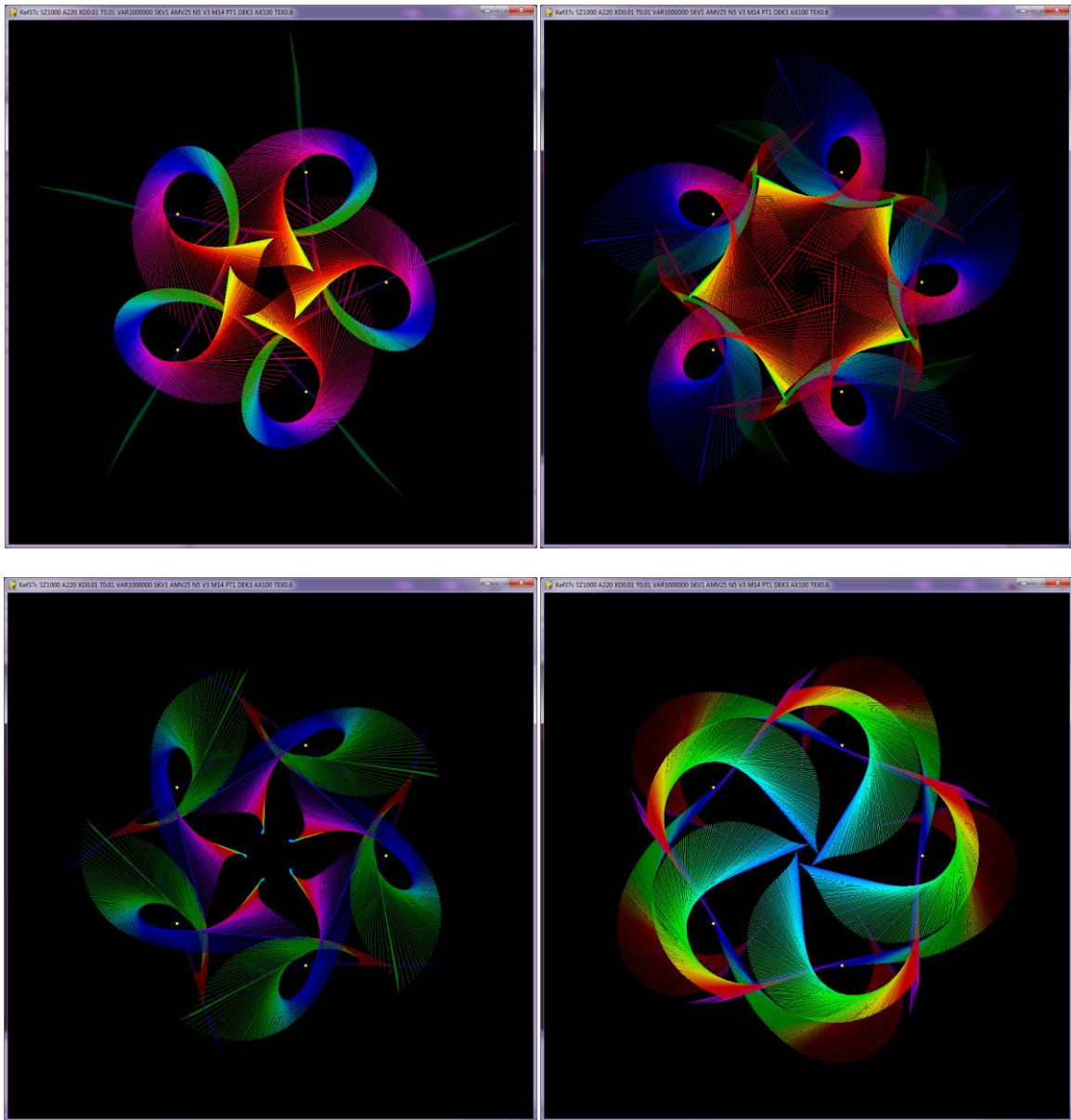
p.vel=p.vel+p.acc*T #Ypologismos neas taxyhtas
p.posA=p.pos
• p.posB=p.pos+p.vel
p.pos=p.pos+p.vel*T #Ypologismos neas theshs
if p.pos[0]>SZ or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
elif p.pos[1]>SZ or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)

for i in range(N): #Edw ypologizontai oi symmetrikes troxies
ps=Object()
ps.posA=(p.posA-V2D((SZ//2,SZ//2))).kart2pol() #ps.posA se polikes apo kentro
ps.posA=V2D((ps.posA[0],ps.posA[1]+i*2*pi/N)).pol2kart() #Athroish i gwniwn
ps.posA=V2D((ps.posA[0]+SZ//2,ps.posA[1]+SZ//2)) #ps.posA apo arxh tw aksonwn
• ps.posB=(p.posB-V2D((SZ//2,SZ//2))).kart2pol() #ps.posB se polikes apo kentro
• ps.posB=V2D((ps.posB[0],ps.posB[1]+i*2*pi/N)).pol2kart() #Athroish i gwniwn
• ps.posB=V2D((ps.posB[0]+SZ//2,ps.posB[1]+SZ//2)) #ps.posB apo arxh tw aksonwn
ps.pos=(p.pos-V2D((SZ//2,SZ//2))).kart2pol() #ps.se polikes apo kentro
ps.pos=V2D((ps.pos[0],ps.pos[1]+i*2*pi/N)).pol2kart() #Athroish i gwniwn
ps.pos=V2D((ps.pos[0]+SZ//2,ps.pos[1]+SZ//2)) #ps.pos apo arxh tw aksonwn
• pygame.draw.line(epif,xrwna(c),(ps.posB[0],ps.posB[1]),(ps.pos[0],ps.pos[1]),PT)
s.planhtes.append(ps)

c=c+TEX #Xrhstimeyei ston xrwmatismo ths poreias, TEX einai taxyhta enallagis xrwmatos
k=k+1 #Efe komhth
if k>DEK: k=0; epif.blit(maska,(0,0)) #DEK einai h diarkeia efe komhth
time.sleep(XD) #Xronikh diakoph
pygame.display.update()

```





ΚΕΦΑΛΑΙΟ 38, Ζεύγη τροχιών

Καταρχήν καλό είναι να εμπλουτίσουμε τα αντικείμενα διανυσμάτων με επιπλέον δυνατότητες, κάποιες από τις οποίες μας χρειάζονται τώρα. Προσθέτουμε λοιπόν στο αρχείο `defClass.py` τέσσερα υποπρογράμματα όπως φαίνεται παρακάτω.

```
def pol2kart(self): #Metatrepei polikes se kartesianes syntetagmenes
    m=self.data[0]
    return V2D([m*cos(self.data[1]), m*sin(self.data[1])])
def gwnia_x(self): #Epistrefetai h gwnia (0 ews 2*pi) poy sxmatizetai me ton axona x
    if self.data[0]>0 and self.data[1]>0: return atan(self.data[1]/self.data[0])
    elif self.data[0]<0 and self.data[1]>0: return pi+atan(self.data[1]/self.data[0])
    elif self.data[0]<0 and self.data[1]<0: return pi+atan(self.data[1]/self.data[0])
    elif self.data[0]>0 and self.data[1]<0: return 2*pi+atan(self.data[1]/self.data[0])
    elif self.data[0]==0 and self.data[1]>0: return pi/2
    elif self.data[0]==0 and self.data[1]<0: return 3*pi/2
    elif self.data[0]>0 and self.data[1]==0: return 0
    elif self.data[0]<0 and self.data[1]==0: return pi
    elif self.data[0]==0 and self.data[1]==0: return 0
def gwnia(self,other): #Epistrefetai h gwnia poy sxmatizetai me to dianysma other
    return other.gwnia_x()-self.gwnia_x()
def symm(self,ka): #Epistrefetai to kathreptismo dianysma ston kathrepth ka
    g=ka.gwnia(self)
    dpolar=self.kart2pol()
    dpolar[1]=dpolar[1]-2*g
    return dpolar.pol2kart()
def provolih(self,other): #Epistrefetai to dianysma provolihs toy self panw sto other
    return other.monadiaio() * self.metro()*cos(self.gwnia(other))
```

Το `gwnia_x` επιστρέφει τη γωνία που σχηματίζεται με τον άξονα x. Το `gwnia` επιστρέφει τη γωνία που σχηματίζεται με το όρισμα. Το `symm` επιστρέφει το καθρεπτισμένο διάνυσμα αν θεωρήσουμε ως καθρέπτη το διάνυσμα `ka`. Το `provolh` επιστρέφει το διάνυσμα προβολής επάνω στο διάνυσμα `other`. Όλα αυτά τα υποπρογράμματα εφαρμόζονται σε διανύσματα και καλό είναι να υπάρχουν στην εργαλειοθήκη μας. Η αναλυτική επεξήγησή τους ξεφεύγει από τα όρια του βιβλίου αυτού, και αρκεί να γνωρίζει κανείς πώς εφαρμόζονται σε διανύσματα.

Στα προηγούμενα παραδείγματα υπάρχει συμμετρία με βάση την περιστροφή. Μπορούμε να προσδώσουμε επιπλέον συμμετρικότητα υπολογίζοντας όχι μία αλλά δύο αρχικές τροχιές καθρεπτισμένες μεταξύ τους καθώς προσεγγίζουν έναν ήλιο. Σε σχέση με το πρόγραμμα `7Kallitexnies37.py`, στο σημείο του `Loop` όπου υπολογίζονταν οι συμμετρικές τροχιές, τροποποιούμε ως εξής:

```
p.pos=p.pos+p.vel*T #Ypologismos neas theshs
if p.pos[0]>SZ or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
elif p.pos[1]>SZ or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
ka=s.hlio[0].pos-V2D([SZ//2,SZ//2]) #kathrepths
pk.pos=(p.pos-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh planhth-kathrepth

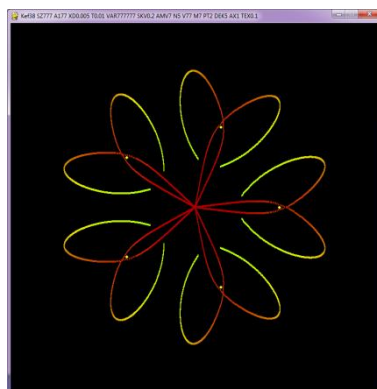
for i in range(N): #Edw ypologizontai oi symmetrikes troxies
    ps=Object()
    ps.pos=(p.pos-V2D([SZ//2,SZ//2])).kart2pol() #p.pos se polikes apo kentro
    ps.pos=V2D([ps.pos[0],ps.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    ps.pos=V2D([ps.pos[0]+SZ//2,ps.pos[1]+SZ//2]) #ps.pos apo arxh twn aksonwn
    pygame.draw.circle(epif,xrwma(c),(int(ps.pos[0]),int(ps.pos[1])),PT//2)
    s.planhtes.append(ps)
    pks=Object()
    pks.pos=(pk.pos-V2D([SZ//2,SZ//2])).kart2pol() #pk.pos se polikes apo kentro
    pks.pos=V2D([pks.pos[0],pks.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    pks.pos=V2D([pks.pos[0]+SZ//2,pks.pos[1]+SZ//2]) #pks.pos apo arxh twn aksonwn
    pygame.draw.circle(epif,xrwma(c),(int(pks.pos[0]),int(pks.pos[1])),PT//2)
    s.planhtes.append(pks)

c=c+TEX #Xrhstimeyi ston xrwmatismo ths poreias, TEX einai taxythta enallagis xrwmatos
k=k+1 #Efe komhth
if k>DEK: k=0; epif.blit(maska,(0,0)) #DEK einai h diarkia efe komhth
time.sleep(XD) #Xronikh diakoph
pygame.display.update()
```

Αλλά και στην αρχή, στο σημείο που δημιουργήσαμε τον πλανήτη `p`, προσθέτουμε μια γραμμή κώδικα για τη δημιουργία πλανήτη-καθρέπτη `pk` του αρχικού πλανήτη `p`, όπως φαίνεται παρακάτω.

```
p=Object() #Dhmioyrgia enos planhth
p.pos=V2D([SZ//2,SZ//2]) #Arxikh thesh planhth
p.vel=V2D([V,2*pi/N+2*pi*M/360]).pol2kart() #Arxikh taxythta planhth
pk=Object() #Dhmioyrgia planhth-kathrepth
```

Αποθηκεύουμε ως `7Kallitexnies38.py` και εκτελούμε.



Παρατηρούμε ότι επετεύχθη το ζητούμενο και κάθε ήλιος προσεγγίζεται πλέον από δύο τροχιές, προσδίδοντας συνολικά μια αίσθηση ισορροπίας.

Μετά τον υπολογισμό της θέσης `p.pos`, για να υπολογιστεί η θέση του πλανήτη-καθρέπτη `pk.pos`, αρχικά η εντολή `ka=s.hlio[0].pos-V2D([SZ//2,SZ//2])` υπολογίζει το

διάνυσμα **ka** που θα παίξει το ρόλο του καθρέπτη. Πρόκειται για την ευθεία που ενώνει τον πρώτο ήλιο με το κέντρο του παραθύρου. Στη συνέχεια, η εντολή **pk.pos=(p.pos-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2])** χρησιμοποιώντας το υποπρόγραμμα **symm**, υπολογίζει τη θέση του πλανήτη καθρέπτη βάση του **p.pos** και του **ka**. Γενικά, το υποπρόγραμμα **symm** δέχεται πριν την τελεία το διάνυσμα που είναι να καθρεπτιστεί και μετά την τελεία σε παρενθέσεις το διάνυσμα που θα παίξει το ρόλο καθρέπτη. Επειδή θέλουμε ως αρχή του διανύσματος θέσης την αρχή των αξόνων, προσθέτουμε τη παράσταση **V2D([SZ//2,SZ//2])**.

Μέσα στη **for** που ακολουθεί δεν υπολογίζονται μόνο οι συμμετρικές θέσεις του **p.pos** αλλά και οι συμμετρικές θέσεις του **pk.pos**.

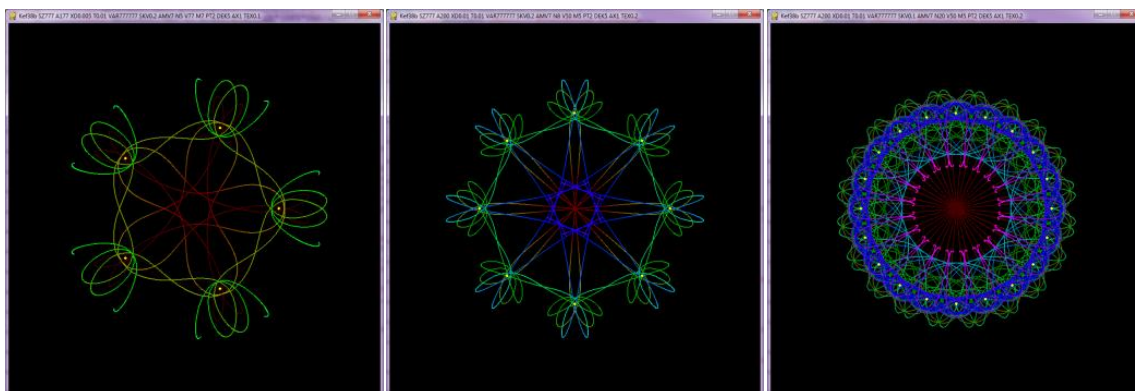
Εάν επιθυμούμε το αντίστοιχο του 7Kallitexnies37b.py, δηλαδή οι τροχιές να σχηματίζονται αντί από κυκλικά στίγματα, από μικρότατα ευθύγραμμα τμήματα τροποποιούμε ως εξής:

```
p.vel=p.vel+p.acc*T #Ypologismos neas taxyhtas
• p.posA=p.pos #Thesh planhth prin
p.pos=p.pos+p.vel*T #Thesh planhth meta
if p.pos[0]>SZ or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
elif p.pos[1]>SZ or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
ka=s.hlio[0].pos-V2D([SZ//2,SZ//2]) #kathrephts
• pk.posA=(p.posA-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh planhth-kathrepht prin
pk.pos=(p.pos-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh planhth-kathrepht meta

for i in range(N): #Edw ypologizontai oi symmetrikes troxies
    ps=Object()
    • ps.posA=(p.posA-V2D([SZ//2,SZ//2])).kart2pol() #p.posA se polikes apo kentro
    • ps.posA=V2D([ps.posA[0],ps.posA[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    • ps.posA=V2D([ps.posA[0]+SZ//2,ps.posA[1]+SZ//2]) #ps.posA apo arxh tw n akswnwn
    ps.pos=(p.pos-V2D([SZ//2,SZ//2])).kart2pol() #p.pos se polikes apo kentro
    ps.pos=V2D([ps.pos[0],ps.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    ps.pos=V2D([ps.pos[0]+SZ//2,ps.pos[1]+SZ//2]) #ps.pos apo arxh tw n akswnwn
    • pygame.draw.line(epif,xrwma(c),(ps.posA[0],ps.posA[1]),(ps.pos[0],ps.pos[1]),PT)
    s.planhtes.append(ps)
    pks=Object()
    • pks.posA=(pk.posA-V2D([SZ//2,SZ//2])).kart2pol() #pk.posA se polikes apo kentro
    • pks.posA=V2D([pks.posA[0],pks.posA[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    • pks.posA=V2D([pks.posA[0]+SZ//2,pks.posA[1]+SZ//2]) #pks.posA apo arxh tw n akswnwn
    pks.pos=(pk.pos-V2D([SZ//2,SZ//2])).kart2pol() #pk.pos se polikes apo kentro
    pks.pos=V2D([pks.pos[0],pks.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    pks.pos=V2D([pks.pos[0]+SZ//2,pks.pos[1]+SZ//2]) #pks.pos apo arxh tw n akswnwn
    • pygame.draw.line(epif,xrwma(c),(pks.posA[0],pks.posA[1]),(pks.pos[0],pks.pos[1]),PT)
    s.planhtes.append(pks)

c=c+TEX #Xrhstimeyei ston xrwmatismo ths poreias, TEX einai taxyhta enallagis xrwmatos
k=k+1 #Efe komhth
if k>DEK: k=0; epif.blit(maska,(0,0)) #DEK einai h diarkia efe komhth
time.sleep(XD) #Xronikh diakoph
pygame.display.update()
```

Αποθηκεύουμε ως 7Kallitexnies38b.py και δοκιμάζουμε εκτελέσεις με διαφορετικές παραμέτρους.



Εάν επιθυμούμε το αντίστοιχο του 7Kallitexnies37c.py, κάνουμε τις τροποποιήσεις του παρακάτω κώδικα και αποθηκεύουμε ως 7Kallitexnies38c.py.


```

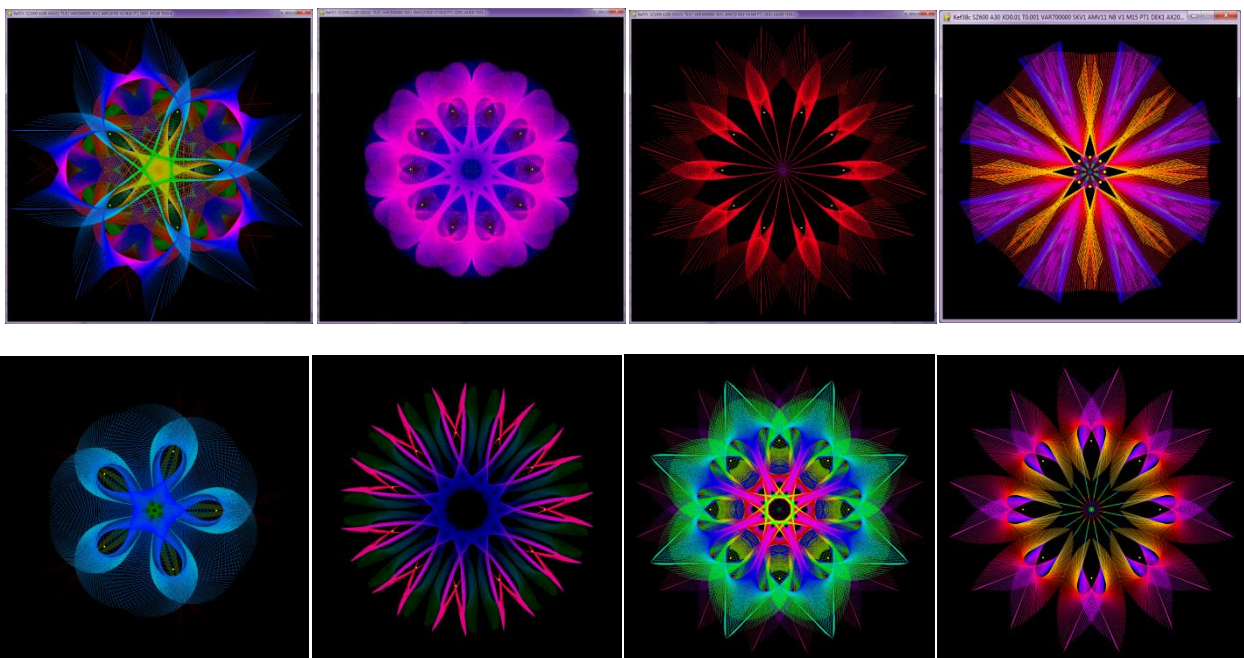
p.vel=p.vel+p.acc*T #Ypologismos neas taxyhtas
p.posA=p.pos #Thesh planhth prin
p.posB=p.pos+p.vel #Thesh p.pos+p.vel
p.pos=p.pos+p.vel*T #Thesh planhth meta
if p.pos[0]>SZ or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
elif p.pos[1]>SZ or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
ka=s.hlio[0].pos-V2D([SZ//2,SZ//2]) #kathrepths
pk.posA=(p.posA-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh planhth-kathrepth prin
pk.posB=(p.posB-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh kathrepth p.pos+p.vel
pk.pos=(p.pos-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh planhth-kathrepth meta

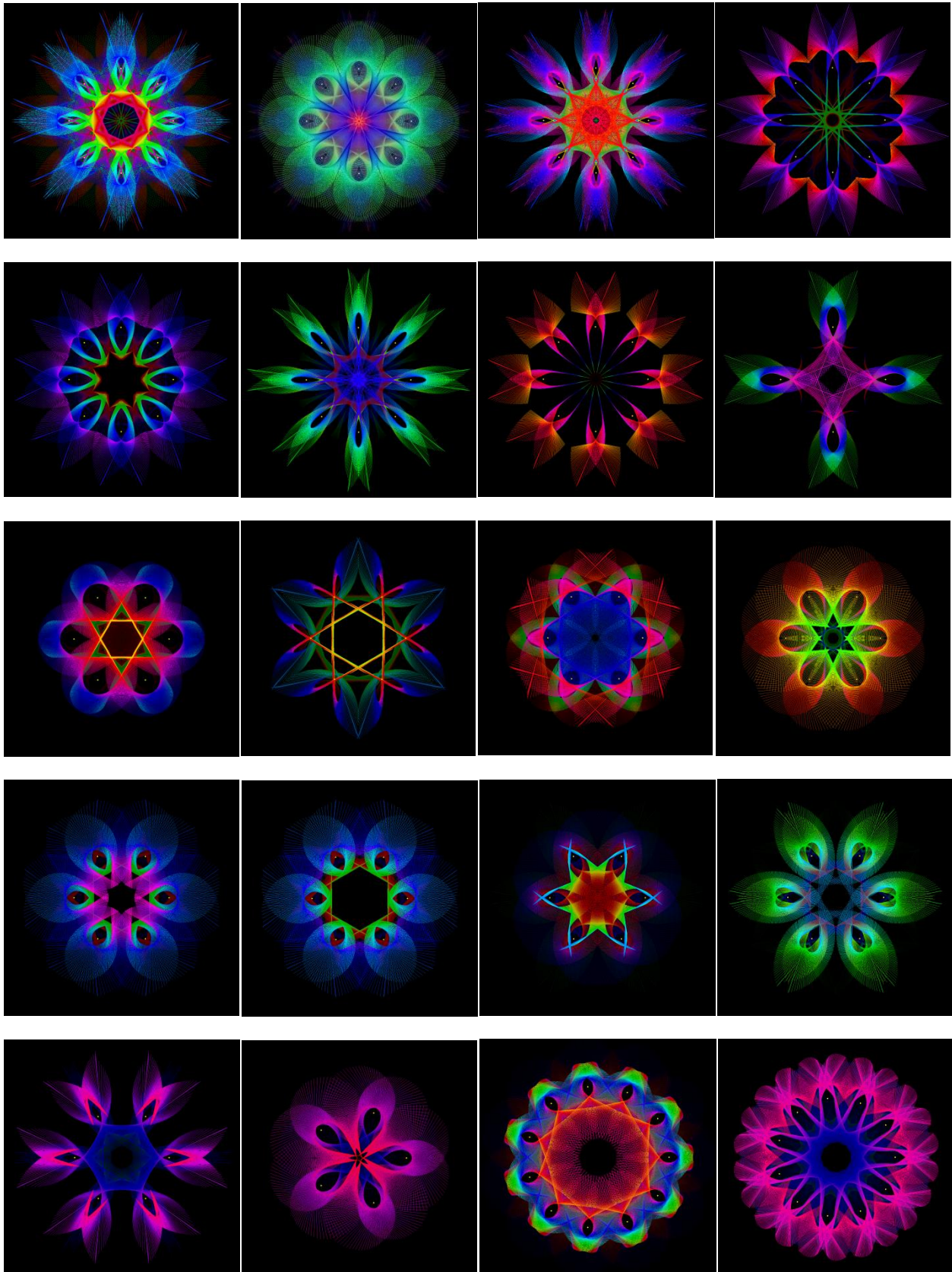
for i in range(N): #Edw ypologizontai oi symmetrikes troxies
    ps=Object()
    ps.posA=(p.posA-V2D([SZ//2,SZ//2])).kart2pol() #p.posA se polikes apo kentro
    ps.posA=V2D([ps.posA[0],ps.posA[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    ps.posA=V2D([ps.posA[0]+SZ//2,ps.posA[1]+SZ//2]) #ps.posA apo arxh twn aksonwn
    ps.posB=(p.posB-V2D([SZ//2,SZ//2])).kart2pol() #p.posB se polikes apo kentro
    ps.posB=V2D([ps.posB[0],ps.posB[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    ps.posB=V2D([ps.posB[0]+SZ//2,ps.posB[1]+SZ//2]) #ps.posB apo arxh twn aksonwn
    ps.pos=(p.pos-V2D([SZ//2,SZ//2])).kart2pol() #p.pos se polikes apo kentro
    ps.pos=V2D([ps.pos[0],ps.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    ps.pos=V2D([ps.pos[0]+SZ//2,ps.pos[1]+SZ//2]) #ps.pos apo arxh twn aksonwn
    pygame.draw.line(epif,xrwma(c),(ps.posB[0],ps.posB[1]),(ps.pos[0],ps.pos[1]),PT)
    s.planhtes.append(ps)
    pks=Object()
    pks.posA=(pk.posA-V2D([SZ//2,SZ//2])).kart2pol() #pk.posA se polikes apo kentro
    pks.posA=V2D([pks.posA[0],pks.posA[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    pks.posA=V2D([pks.posA[0]+SZ//2,pks.posA[1]+SZ//2]) #pks.posA apo arxh twn aksonwn
    pks.posB=(pk.posB-V2D([SZ//2,SZ//2])).kart2pol() #pk.posB se polikes apo kentro
    pks.posB=V2D([pks.posB[0],pks.posB[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    pks.posB=V2D([pks.posB[0]+SZ//2,pks.posB[1]+SZ//2]) #pks.posB apo arxh twn aksonwn
    pks.pos=(pk.pos-V2D([SZ//2,SZ//2])).kart2pol() #pk.pos se polikes apo kentro
    pks.pos=V2D([pks.pos[0],pks.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    pks.pos=V2D([pks.pos[0]+SZ//2,pks.pos[1]+SZ//2]) #pks.pos apo arxh twn aksonwn
    pygame.draw.line(epif,xrwma(c),(pks.posB[0],pks.posB[1]),(pks.pos[0],pks.pos[1]),PT)
    s.planhtes.append(pks)

c=c+TEX #Xrhsimeyei ston xrwmatismo ths poreias, TEX einai taxyhtta enallagis xrwmatos
k=k+1 #Efe komhth
if k>DEK: k=0; epif.blit(maska,(0,0)) #DEK einai h diarkeia efe komhth
time.sleep(XD) #Xronikh diakoph
pygame.display.update()

```

Η θέση **p.posA** δεν χρησιμοποιείται σε αυτό το πρόγραμμα και άρα οι αντίστοιχες γραμμές που έχουν **ps.posA** και **pks.posA** μπορούν και να απουσιάζουν. Δοκιμάζουμε το παραπάνω πρόγραμμα με διάφορες παραμέτρους.





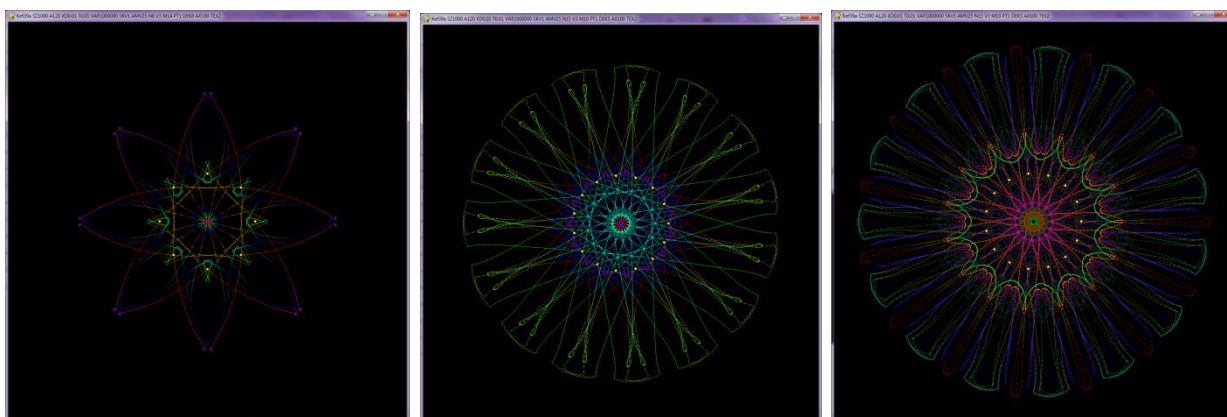
Στα προγράμματα 7Kallitexnies37b.py, 7Kallitexnies37c.py, 7Kallitexnies38b.py και 7Kallitexnies38c.py καλό είναι να δοκιμάσει κανείς την εντολή `pygame.draw.aaline` (εξηγήθηκε στο κεφάλαιο 15) αντί για την `pygame.draw.line`. Η ιδιότητα `antialiasing` θα προκαλέσει να αποδοθούν καλύτερα οι γραμμές. Όμως, η ιδιότητα εφαρμόζεται μόνο σε σχεδιασμό γραμμών με πάχος 1 πίξελ.

ΚΕΦΑΛΑΙΟ 39, Δημιουργικότητα

Μπορεί κανείς πλέον εύκολα και με τυποποιημένες διαδικασίες τροποποίησης του κώδικα, να δοκιμάσει ότι θεωρεί ότι πιθανόν να έχει ενδιαφέρον και δημιουργικό αποτέλεσμα. Για παράδειγμα, στον παρακάτω κώδικα σχεδιάζονται γραμμές από το σημείο `p.posC` προς το σημείο `p.posB`, όπου το `p.posC` αναφέρεται στην αρχική θέση `p.pos + p.vel`, ενώ το `p.posB` αναφέρεται στην επόμενη θέση `p.pos + p.vel`. Όπως και στο πρόγραμμα 7Kallitexnies37c.py, για να μην υπολογίζονται θέσεις εκτός παραθύρου, συνιστάται μεγάλο `AMV`, μικρό `N` ή δοκιμές με άλλες αλλαγές παραμέτρων με αντίστοιχο αποτέλεσμα.

```
else: p.acc=p.acc+ph*(VAR/apostash**3)
p.posC=p.pos+p.vel #Thesh p.pos+p.vel prin
p.vel=p.vel+p.acc*T #Ypologismos neas taxyhttas
p.posA=p.pos #Thesh planhth prin
p.posB=p.pos+p.vel #Thesh p.pos+p.vel meta
p.pos=p.pos+p.vel*T #Thesh planhth meta
if p.pos[0]>SZ or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
elif p.pos[1]>SZ or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
ka=s.hlioi[0].pos-V2D([SZ//2,SZ//2]) #kathrepths
pk.posA=(p.posA-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh planhth-kathrepth prin
pk.pos=(p.pos-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh planhth-kathrepth meta
pk.posC=(p.posC-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh kathrepth p.pos+p.vel prin
pk.posB=(p.posB-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh kathrepth p.pos+p.vel meta
for i in range(N): #Edw ypologizontai oi symmetrikes troxies
    ps=Object()
    ps.posA=(p.posA-V2D([SZ//2,SZ//2])).kart2pol() #p.posA se polikes apo kentro
    ps.posA=V2D([ps.posA[0],ps.posA[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    ps.posA=V2D([ps.posA[0]+SZ//2,ps.posA[1]+SZ//2]) #ps.posA apo arxh tw n aksonwn
    ps.posB=(p.posB-V2D([SZ//2,SZ//2])).kart2pol() #p.posB se polikes apo kentro
    ps.posB=V2D([ps.posB[0],ps.posB[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    ps.posB=V2D([ps.posB[0]+SZ//2,ps.posB[1]+SZ//2]) #ps.posB apo arxh tw n aksonwn
    ps.posC=(p.posC-V2D([SZ//2,SZ//2])).kart2pol() #p.posC se polikes apo kentro
    ps.posC=V2D([ps.posC[0],ps.posC[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    ps.posC=V2D([ps.posC[0]+SZ//2,ps.posC[1]+SZ//2]) #ps.posC apo arxh tw n aksonwn
    ps.pos=(p.pos-V2D([SZ//2,SZ//2])).kart2pol() #p.pos se polikes apo kentro
    ps.pos=V2D([ps.pos[0],ps.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    ps.pos=V2D([ps.pos[0]+SZ//2,ps.pos[1]+SZ//2]) #ps.pos apo arxh tw n aksonwn
    pygame.draw.aaline(epif,xrwma(c),(ps.posC[0],ps.posC[1]),(ps.posB[0],ps.posB[1]),PT)
    s.planhtes.append(ps)
    pks=Object()
    pks.posA=(pk.posA-V2D([SZ//2,SZ//2])).kart2pol() #pk.posA se polikes apo kentro
    pks.posA=V2D([pks.posA[0],pks.posA[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    pks.posA=V2D([pks.posA[0]+SZ//2,pks.posA[1]+SZ//2]) #pks.posA apo arxh tw n aksonwn
    pks.posB=(pk.posB-V2D([SZ//2,SZ//2])).kart2pol() #pk.posB se polikes apo kentro
    pks.posB=V2D([pks.posB[0],pks.posB[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    pks.posB=V2D([pks.posB[0]+SZ//2,pks.posB[1]+SZ//2]) #pks.posB apo arxh tw n aksonwn
    pks.posC=(pk.posC-V2D([SZ//2,SZ//2])).kart2pol() #pk.posC se polikes apo kentro
    pks.posC=V2D([pks.posC[0],pks.posC[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    pks.posC=V2D([pks.posC[0]+SZ//2,pks.posC[1]+SZ//2]) #pks.posC apo arxh tw n aksonwn
    pks.pos=(pk.pos-V2D([SZ//2,SZ//2])).kart2pol() #pk.pos se polikes apo kentro
    pks.pos=V2D([pks.pos[0],pks.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
    pks.pos=V2D([pks.pos[0]+SZ//2,pks.pos[1]+SZ//2]) #pks.pos apo arxh tw n aksonwn
    pygame.draw.aaline(epif,xrwma(c),(pks.posC[0],pks.posC[1]),(pks.posB[0],pks.posB[1]),PT)
    s.planhtes.append(pks)
```

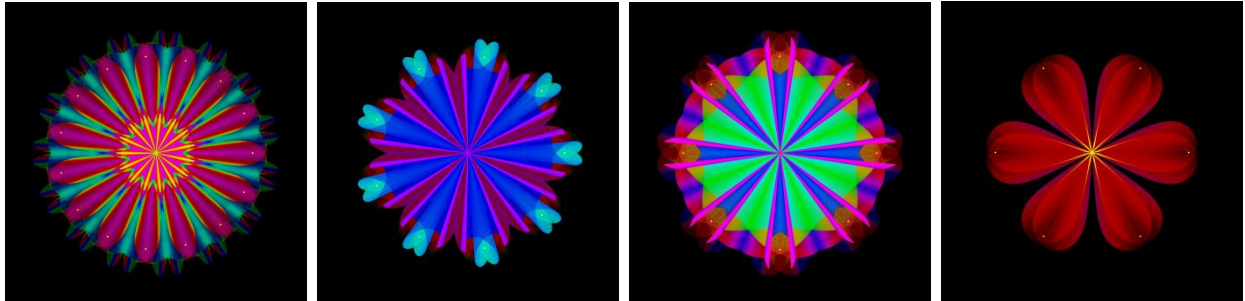
Αποθηκεύουμε π.χ. ως 7Kallitexnies39a.py και τρέχουμε.



Στο προηγούμενο πρόγραμμα αντικαθιστούμε μόνο τις δύο εντολές `pygame.draw` ως εξής:

```
pygame.draw.aaline(epif,xrwma(c),(SZ//2,SZ//2),(ps.pos[0],ps.pos[1]),PT)
pygame.draw.aaline(epif,xrwma(c),(SZ//2,SZ//2),(pks.pos[0],pks.pos[1]),PT)
```

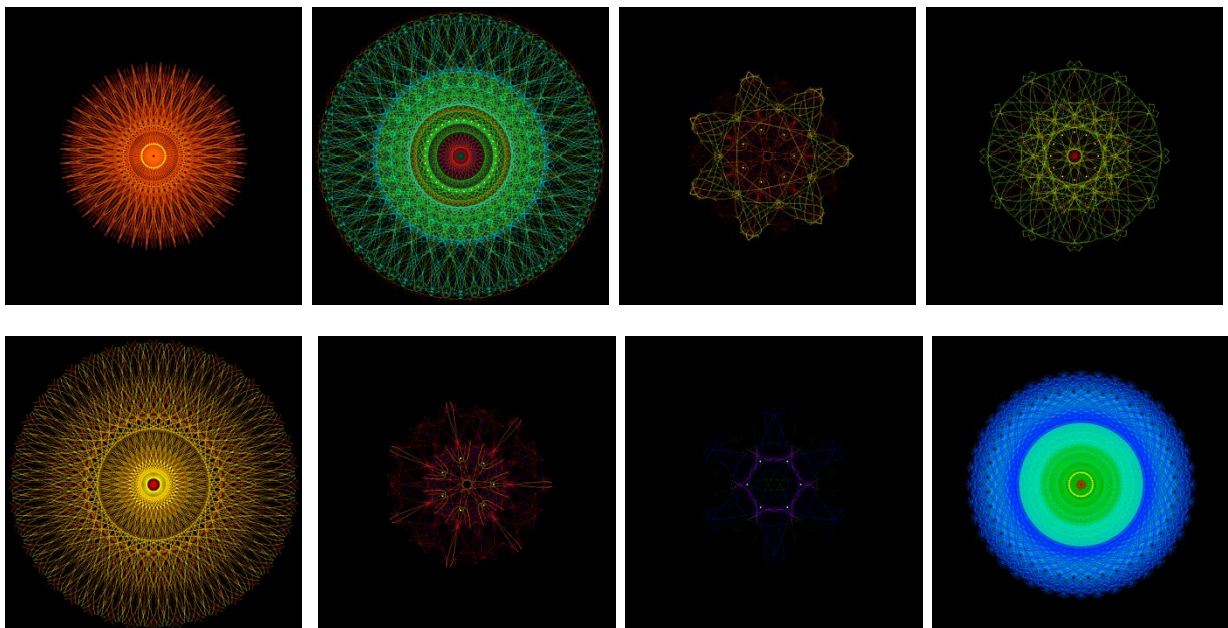
Δηλαδή, ορίζουμε ως σταθερό άκρο των σχεδιαζόμενων γραμμών το κέντρο του παραθύρου $(SZ//2, SZ//2)$, ενώ ως δεύτερο άκρο ορίζουμε το σημείο `p.pos`. Αποθηκεύουμε π.χ. ως 7Kallitexnies39b.py και τρέχουμε. Προκύπτουν επίσης πολύ ενδιαφέρουσες δημιουργίες.



Στο προηγούμενο πρόγραμμα αντικαθιστούμε μόνο τις δύο εντολές `pygame.draw` ως εξής:

```
pygame.draw.aaline(epif,xrwma(c),(ps.posB[0],ps.posB[1]),(ps.posC[0],ps.posC[1]),PT)
pygame.draw.aaline(epif,xrwma(c),(pks.posB[0],pks.posB[1]),(pks.posC[0],pks.posC[1]),PT)
```

Δηλαδή, ορίζουμε ως πρώτο άκρο των σχεδιαζόμενων γραμμών το σημείο `p.posB`, ενώ ως δεύτερο άκρο ορίζουμε το σημείο `p.posC`. Το τι αντιπροσωπεύουν τα `p.posB` και `p.posC` έχει επεξηγηθεί προηγουμένως. Αποθηκεύουμε π.χ. ως 7Kallitexnies39c.py και τρέχουμε. Προκύπτουν και πάλι ενδιαφέρουσες δημιουργίες.



Έχει ενδιαφέρον να ορίσουμε ως πρώτο άκρο των σχεδιαζόμενων γραμμών τα σταθερά σημεία των ήλιων, ενώ ως δεύτερο άκρο να ορίσουμε τα μεταβαλλόμενα σημεία `p.pos`. Κάνουμε τις απαραίτητες προσθήκες κώδικα (βλέπε παρακάτω), ουσιαστικά δημιουργώντας το σημείο του ήλιου `p.posD` και όλα τα σχετικά. Αν και έχουν αλλάξει μόνο λίγες γραμμές, παρατίθεται παρακάτω ολόκληρος ο κώδικας του Loop.

```

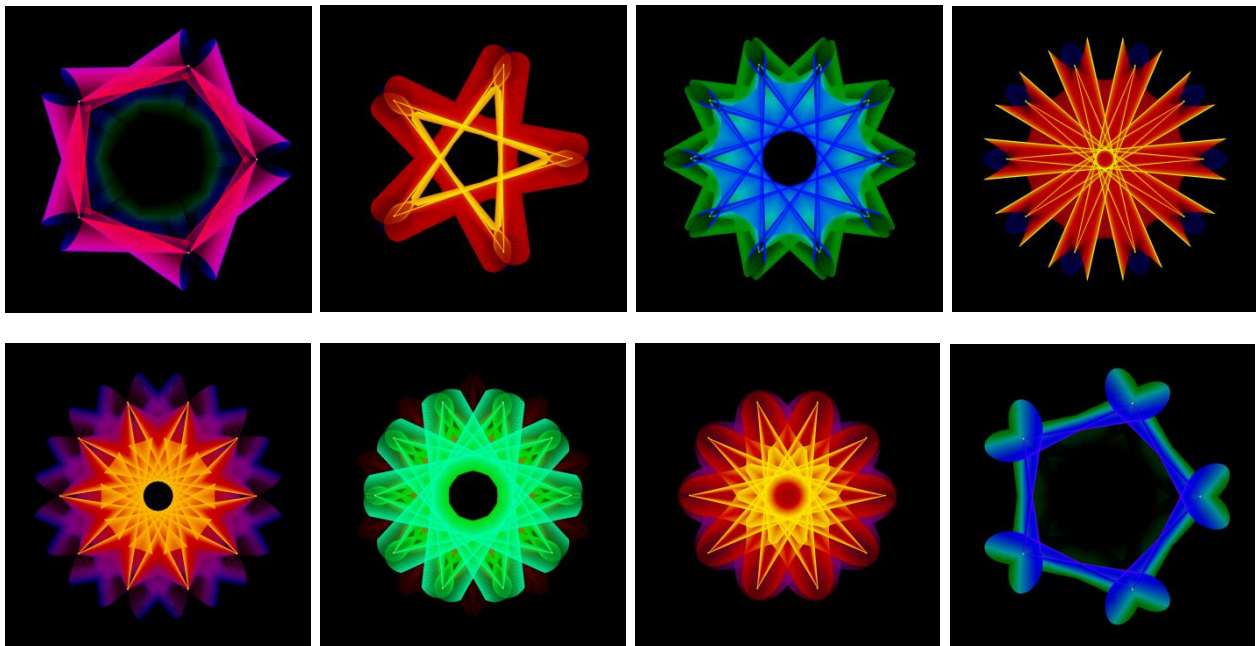
while True: #Loop
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit(); sys.exit()
        elif event.type == pygame.KEYDOWN and event.key == pygame.K_s:
            namebmp='Kef39d SZ'+str(SZ)+' A'+str(A)+' XD'+str(XD)+' T'+str(T)+' VAR'+str(VAR) \
            +' SKV'+str(SKV)+' AMV'+str(AMV)+' N'+str(N)+' V'+str(V)+' M'+str(M)+' PT'+str(PT) \
            +' DEK'+str(DEK)+' AX'+str(AX)+' TEX'+str(TEX)+' sec'+str(int(c*XD/TEX))+'.bmp'
            pygame.image.save(epif,namebmp)
    for h in s.hlioi:
        pygame.draw.circle(epif, (255,255,0), (int(h.pos[0]),int(h.pos[1])),3)

    p.acc=V2D([0,0])
    for h in s.hlioi:
        ph=h.pos-p.pos #Dianysma apostashs planhth-hlioy
        apostash=ph.metro() #Metro dianysmatos apostashs
        if apostash<=AMV: p.acc=p.acc+V2D([0,0]) #Ypologismos epitaxyynshs
        elif apostash<=AMV*2: p.acc=p.acc+ph*(SKV*VAR/apostash**3)
        else: p.acc=p.acc+ph*(VAR/apostash**3)
    p.posC=p.pos+p.vel #Thesh p.pos+p.vel prin
    p.vel=p.vel+p.acc*T #Ypologismos neas taxytthas
    p.posA=p.pos #Thesh planhth prin
    p.posB=p.pos+p.vel #Thesh p.pos+p.vel meta
    p.pos=p.pos+p.vel*T #Thesh planhth meta
    p.posD=s.hlioi[0].pos #Thesh prwtoy hlioy
    if p.pos[0]>SZ or p.pos[0]<0: p.vel[0]=p.vel[0]*(-1)
    elif p.pos[1]>SZ or p.pos[1]<0: p.vel[1]=p.vel[1]*(-1)
    ka=s.hlioi[0]
    ka=V2D([SZ//2,SZ//2]) #kathrepths
    pk.posA=(p.posA-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh planhth-kathrepth prin
    pk.pos=(p.pos-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh planhth-kathrepth meta
    pk.posC=(p.posC-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh kathrepth p.pos+p.vel prin
    pk.posB=(p.posB-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh kathrepth p.pos+p.vel meta
    pk.posD=(p.posD-V2D([SZ//2,SZ//2])).symm(ka)+V2D([SZ//2,SZ//2]) #Thesh kathrepth prwtoy hlioy
    for i in range(N): #Edw ypologizontai oi symmetrikes troxies
        ps=Object()
        ps.posA=(p.posA-V2D([SZ//2,SZ//2])).kart2pol() #p.posA se polikes apo kentro
        ps.posA=V2D([ps.posA[0],ps.posA[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        ps.posA=V2D([ps.posA[0]+SZ//2,ps.posA[1]+SZ//2]) #ps.posA apo arxh tw n aksonwn
        ps.posB=(p.posB-V2D([SZ//2,SZ//2])).kart2pol() #p.posB se polikes apo kentro
        ps.posB=V2D([ps.posB[0],ps.posB[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        ps.posB=V2D([ps.posB[0]+SZ//2,ps.posB[1]+SZ//2]) #ps.posB apo arxh tw n aksonwn
        ps.posC=(p.posC-V2D([SZ//2,SZ//2])).kart2pol() #p.posC se polikes apo kentro
        ps.posC=V2D([ps.posC[0],ps.posC[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        ps.posC=V2D([ps.posC[0]+SZ//2,ps.posC[1]+SZ//2]) #ps.posC apo arxh tw n aksonwn
        ps.posD=(p.posD-V2D([SZ//2,SZ//2])).kart2pol() #p.posD se polikes apo kentro
        ps.posD=V2D([ps.posD[0],ps.posD[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        ps.posD=V2D([ps.posD[0]+SZ//2,ps.posD[1]+SZ//2]) #ps.posD apo arxh tw n aksonwn
        ps.pos=(p.pos-V2D([SZ//2,SZ//2])).kart2pol() #p.pos se polikes apo kentro
        ps.pos=V2D([ps.pos[0],ps.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        ps.pos=V2D([ps.pos[0]+SZ//2,ps.pos[1]+SZ//2]) #ps.pos apo arxh tw n aksonwn
        pygame.draw.aaline(epif,xrwma(c), (ps.pos[0],ps.pos[1]), (ps.posD[0],ps.posD[1]),PT)
        s.planhtes.append(ps)
        pks=Object()
        pks.posA=(pk.posA-V2D([SZ//2,SZ//2])).kart2pol() #pk.posA se polikes apo kentro
        pks.posA=V2D([pks.posA[0],pks.posA[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        pks.posA=V2D([pks.posA[0]+SZ//2,pks.posA[1]+SZ//2]) #pks.posA apo arxh tw n aksonwn
        pks.posB=(pk.posB-V2D([SZ//2,SZ//2])).kart2pol() #pk.posB se polikes apo kentro
        pks.posB=V2D([pks.posB[0],pks.posB[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        pks.posB=V2D([pks.posB[0]+SZ//2,pks.posB[1]+SZ//2]) #pks.posB apo arxh tw n aksonwn
        pks.posC=(pk.posC-V2D([SZ//2,SZ//2])).kart2pol() #pk.posC se polikes apo kentro
        pks.posC=V2D([pks.posC[0],pks.posC[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        pks.posC=V2D([pks.posC[0]+SZ//2,pks.posC[1]+SZ//2]) #pks.posC apo arxh tw n aksonwn
        pks.posD=(pk.posD-V2D([SZ//2,SZ//2])).kart2pol() #pk.posD se polikes apo kentro
        pks.posD=V2D([pks.posD[0],pks.posD[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        pks.posD=V2D([pks.posD[0]+SZ//2,pks.posD[1]+SZ//2]) #pks.posD apo arxh tw n aksonwn
        pks.pos=(pk.pos-V2D([SZ//2,SZ//2])).kart2pol() #pk.pos se polikes apo kentro
        pks.pos=V2D([pks.pos[0],pks.pos[1]+i*2*pi/N]).pol2kart() #Athroish i gwniwn
        pks.pos=V2D([pks.pos[0]+SZ//2,pks.pos[1]+SZ//2]) #pks.pos apo arxh tw n aksonwn
        pygame.draw.aaline(epif,xrwma(c), (pks.pos[0],pks.pos[1]), (pks.posD[0],pks.posD[1]),PT)
        s.planhtes.append(pks)

    c=c+TEX #Xrhstimeyei ston xrwmatismo ths poreias, TEX einai taxyttha enallagis xrwmatos
    k=k+1 #Efe komhth
    if k>DEK: k=0; epif.blit(maska, (0,0)) #DEK einai h diarkia efe komhth
    time.sleep(XD) #Xronikh diakoph
    pygame.display.update()

```

Αποθηκεύουμε π.χ. ως 7Kallitexnies39d.py και τρέχουμε. Προκύπτουν άλλου είδους άπειρες ενδιαφέρουσες δημιουργίες καθώς δοκιμάζουμε διάφορες παραμέτρους.



ΚΕΦΑΛΑΙΟ 40, Σύνθεση

Μπορεί κανείς εύκολα να ενσωματώσει τα προγράμματα των κεφαλαίων 37, 38 και 39 σε ένα πρόγραμμα. Προς αυτό το σκοπό, συνιστάται στο σημείο των σταθερών να δημιουργηθεί μια αλφαριθμητική σταθερά επιπλέον, π.χ. ονόματος `EIDOS` όπως φαίνεται παρακάτω. Αυτή θα δέχεται ως τιμή το είδος της καλλιτεχνίας, δηλαδή π.χ. για τις δημιουργίες του κεφαλαίου 38c θα ορίζεται ως τιμή το 38c. Λίγο πιο κάτω, πρέπει να γίνει βέβαια μια μικρή αλλαγή στην εντολή `pygame.display.set_caption`.

```
AX=0          #Arxikos xrwmatismos
TEX=0.05      #Taxythta enallaghs xrwmatos
EIDOS='38c'   #Eidos kallitexnias 37,37b,37c,38,38b,38c,39a,39b,39c,39d

pygame.init()
epif=pygame.display.set_mode((SZ,SZ))
pygame.display.set_caption('Kef'+EIDOS+' SZ'+str(SZ)+' A'+str(A)+' XD'+str(XD)+' T'+str(T)+
    ' VAR'+str(VAR)+' SKV'+str(SKV)+' AMV'+str(AMV)+' N'+str(N)+' V'+str(V)+' M'+str(M)+
    ' PT'+str(PT)+' DEK'+str(DEK)+' AX'+str(AX)+' TEX'+str(TEX) )
```

Το αντίστοιχο πρέπει να γίνει και στο σημείο του κώδικα όπου με το πλήκτρο S γίνεται αποθήκευση, ώστε να ονοματίζεται ορθά η αποθηκευμένη εικόνα.

Παρακάτω, στο σημείο του κώδικα όπου δύο φορές υπήρχε η `pygame.draw.aaline` τώρα πρέπει να προστεθούν πολλαπλές επιλογές.

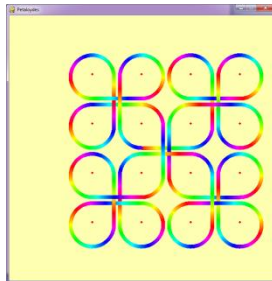
```
ps.pos=v2d((ps.pos[0]+SZ//2,ps.pos[1]+SZ//2)) #ps.pos apo arxh tw n aksonwn
if EIDOS=='37': pygame.draw.circle(epif,xrwma(c),(int(ps.pos[0]),int(ps.pos[1])),PT//2)
elif EIDOS=='37b': pygame.draw.aaline(epif,xrwma(c),(ps.posA[0],ps.posA[1]),(ps.pos[0],ps.pos[1]),PT)
elif EIDOS=='37c': pygame.draw.aaline(epif,xrwma(c),(ps.posB[0],ps.posB[1]),(ps.pos[0],ps.pos[1]),PT)
elif EIDOS=='38': pygame.draw.circle(epif,xrwma(c),(int(ps.pos[0]),int(ps.pos[1])),PT//2)
elif EIDOS=='38b': pygame.draw.aaline(epif,xrwma(c),(ps.posA[0],ps.posA[1]),(ps.pos[0],ps.pos[1]),PT)
elif EIDOS=='38c': pygame.draw.aaline(epif,xrwma(c),(ps.posB[0],ps.posB[1]),(ps.pos[0],ps.pos[1]),PT)
elif EIDOS=='39a': pygame.draw.aaline(epif,xrwma(c),(ps.posC[0],ps.posC[1]),(ps.posB[0],ps.posB[1]),PT)
elif EIDOS=='39b': pygame.draw.aaline(epif,xrwma(c),(SZ//2,SZ//2),(ps.pos[0],ps.pos[1]),PT)
elif EIDOS=='39c': pygame.draw.aaline(epif,xrwma(c),(ps.posB[0],ps.posB[1]),(ps.posC[0],ps.posC[1]),PT)
elif EIDOS=='39d': pygame.draw.aaline(epif,xrwma(c),(ps.pos[0],ps.pos[1]),(ps.posD[0],ps.posD[1]),PT)
elif EIDOS=='40a': pygame.draw.aaline(epif,xrwma(c),(ps.posC[0],ps.posC[1]),(ps.posD[0],ps.posD[1]),PT)
elif EIDOS=='40b': pygame.draw.aaline(epif,xrwma(c),(SZ//2+2*A*cos(c),SZ//2+2*A*sin(c)),(ps.pos[0],ps.pos[1]),PT)
s.planhtes.append(ps)
```

Και λίγο παρακάτω...


```
pks.pos=V2D([pks.pos[0]+SZ//2,pks.pos[1]+SZ//2]) #pks.pos apo arxh tw n aksonwn
if EIDOS=='37': pass
elif EIDOS=='37b': pass
elif EIDOS=='37c': pass
elif EIDOS=='38': pygame.draw.circle(epif,xrwma(c),(int(pks.pos[0]),int(pks.pos[1])),PT//2)
elif EIDOS=='38b': pygame.draw.aaline(epif,xrwma(c),(pks.posA[0],pks.posA[1]),(pks.pos[0],pks.pos[1]),PT)
elif EIDOS=='38c': pygame.draw.aaline(epif,xrwma(c),(pks.posB[0],pks.posB[1]),(pks.pos[0],pks.pos[1]),PT)
elif EIDOS=='39a': pygame.draw.aaline(epif,xrwma(c),(pks.posC[0],pks.posC[1]),(pks.posB[0],pks.posB[1]),PT)
elif EIDOS=='39b': pygame.draw.aaline(epif,xrwma(c),(SZ//2,SZ//2),(pks.pos[0],pks.pos[1]),PT)
elif EIDOS=='39c': pygame.draw.aaline(epif,xrwma(c),(pks.posB[0],pks.posB[1]),(pks.posC[0],pks.posC[1]),PT)
elif EIDOS=='39d': pygame.draw.aaline(epif,xrwma(c),(pks.pos[0],pks.pos[1]),(pks.posD[0],pks.posD[1]),PT)
elif EIDOS=='40a': pygame.draw.aaline(epif,xrwma(c),(pks.posC[0],pks.posC[1]),(pks.posD[0],pks.posD[1]),PT)
elif EIDOS=='40b': pygame.draw.aaline(epif,xrwma(c),(SZ//2+2*A*cos(c),SZ//2+2*A*sin(c)),(pks.pos[0],pks.pos[1]),PT)
s.planhthes.append(ros)
```

Παρατηρεί κανείς ότι έχουν προστεθεί δύο νέα είδη δημιουργιών. Το 40a αφορά γραμμές που ενώνουν το **posC** με το **posD**, ενώ το 40b αφορά γραμμές που εκκινούν από σημεία κύκλου ακτίνας $2 \cdot A$ και τερματίζουν στο σημείο **pos**. Αποθηκεύουμε π.χ. ως 7Kallitexnies40.py. Επιπλέον, στο πρόγραμμα 7Kallitexnies40TELIKO.py υπάρχει η παράμετρος **EIDOSXR** που καθορίζει το είδος του χρωματισμού (xrwma, xrwmaRYMR, xrwmaGCGY, xrwmaBMCB, xrwmaGMRG, xrwmaBRMB, xrwmaRMWYR, xrwmaRMWYWR, xrwmaPrasino, xrwmaKokkino, ή xrwmaMple) και χρησιμοποιείται για αυτό η εντολή **exec**.

ΚΕΦΑΛΑΙΟ 41, Πεταλούδες



Υπάρχει ένα μοτίβο σχεδίου, το οποίο προσωπικά εκτιμώ και που το ονόμασα «Πεταλούδα», λόγω του βασικού συστατικού που περιλαμβάνει και που είναι σχήματος πεταλούδας $\bowtie$.

Γράφουμε τον κώδικα 7Kallitexnies41.py που υπάρχει στην ιστοσελίδα του βιβλίου 7777777.gr και τον τρέχουμε.

Το αποτέλεσμα επιτυγχάνεται με πολλαπλά **for**, τα οποία σχεδιάζουν το καθένα από συγκεκριμένο τμήμα (κυκλικό ή γραμμικό) του τελικού σχεδίου, σε συνδυασμό με τις τεχνικές μεταβολής χρωματισμού που εξηγήθηκαν στα προηγούμενα κεφάλαια.

ΕΠΙΛΟΓΟΣ

Σε αυτό το βιβλίο ουσιαστικά έγινε προσπάθεια να παρουσιαστούν ανεξερεύνητοι μέχρι στιγμής εικαστικοί καλαισθητοί χώροι, αλλά παράλληλα έγινε προσπάθεια να δοθούν και τα κατάλληλα εργαλεία με τα οποία μπορεί κανείς μόνος του πλέον να ανακαλύψει άπειρους άλλους όμορφους χώρους. Χώρους κρυμμένους πίσω από αλγοριθμικά πέπλα, κάτι σαν τους χώρους φράκταλ.

Έπεται συνέχεια του βιβλίου. Όσο υποχωρούν τα νερά της άγνοιας, τόσο αναδύεται μεγαλύτερο το νησί της γνώσης. Και μάλιστα οι ακτές, δηλαδή τα μέτωπα εξερεύνησης πληθαίνουν. Η μεγάλη πρόκληση αφορά ασφαλώς τη μετάβαση από τις δύο στις τρεις διαστάσεις και εκεί πλέον μπορεί κανείς να υποψιαστεί βάσιμα ότι με παρόμοιες αλγοριθμικές τεχνικές και εκτυπωτές τρισδιάστατων αντικειμένων, δημιουργούνται αντικείμενα που δεν έχουν να ζηλέψουν τίποτα από τα φυτά και τα άνθη της γης εξαιρουμένου του αρώματός τους.

έγχρωμο βιβλίο

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ ΒΙΒΛΙΟΥ:

- ΠΡΟΣΕΓΓΙΣΗ ΒΗΜΑ ΠΡΟΣ ΒΗΜΑ
- ΕΠΕΞΗΓΗΣΗ ΟΛΩΝ ΤΩΝ ΣΗΜΕΙΩΝ
- ΕΞΑΙΡΕΤΙΚΑ ΔΗΜΙΟΥΡΓΙΚΗ ΓΝΩΣΗ
- ΔΙΑΘΕΣΙΜΟΣ ΚΩΔΙΚΑΣ
- ΕΚΜΑΘΗΣΗ ΤΗΣ ΓΛΩΣΣΑΣ ΡΥΘΜΟΝ
- ΣΥΝΙΣΤΑΤΑΙ ΚΑΙ ΓΙΑ ΑΡΧΑΡΙΟΥΣ

«Με το πρωτότυπο αυτό βιβλίο “ΚΑΛΛΙΤΕΧΝΙΕΣ ΜΕ ΥΠΟΛΟΓΙΣΤΗ” ο Γιώργος Ψαράκης παρουσιάζει με εξαιρετικό τρόπο ένα πολύ ενδιαφέρον εικαστικό ζήτημα και προσφέρει απίστευτα δημιουργική γνώση!».

e-book

ISBN 978-960-88910-4-3

