Κεφάλαιο 1

Γεωμετρικά Αρχέτυπα και Κατηγορήματα

1.1 Γεωμετρικά Αρχέτυπα

Ξεκινάμε την υλοποίηση της γεωμετρικής βιβλιοθήκης με την κωδικοποίηση στην Python των κλάσεων που αναπαριστούν τις βασικές γεωμετρικές οντότητες που χειρίζονται οι γεωμετρικοί αλγόριθμοι: σημεία, ευθύγραμμα τμήματα και πολύγωνα. Στη συνέχεια κωδικοποιούμε διάφορα κατηγορήματα προσανατολισμού που αποτελούν τα βασικότερα κατηγορήματα της Υπολογιστικής Γεωμετρίας και θα χρησιμοποιηθούν σε μια πληθώρα από γεωμετρικούς αλγόριθμους που θα περιγράψουμε και θα υλοποιήσουμε.

1.1.1 Αναπαράσταση σημείων

Μια κλάση της Python που τα στιγμιότυπά τους αναπαριστούν σημεία στο επίπεδο είναι η κλάση που φαίνεται στο πλαίσιο *κλάση Python* 1.1. Τα στιγμιότυπα της κλάσης έχουν δύο χαρακτηριστικά (attributes), τις συντεταγμένες `x` και `y`. Οι δύο μέθοδοι `from_point2` και `from_tuple` χρησιμοποιούν το ιδίωμα της Python που προσομοιάζει αυτό που σε άλλες γλώσσες προγραμματισμού ονομάζεται υπερφόρτωση του κατασκευαστή (constructor overloading). Με τη χρήση αυτών των μεθόδων μπορούμε να αρχικοποιήσουμε ένα στιγμιότυπο της κλάσης `Point2` από άλλα στιγμιότυπα κλάσεων όπως ένα στιγμιότυπο της ίδιας κλάσης ή ακόμη και από μια πλειάδα που αναπαριστά τις συντεταγμένες του σημείου. Με τη χρήση της ιδιότητας `coordinates` μπορούμε να έχουμε την απευθείας ανάθεση μιας πλειάδας στο στιγμιότυπο της `Point2` με αποτέλεσμα την άμεση ανάθεση των στοιχείων της πλειάδας στις αντίστοιχες συντεταγμένες.

Στο πλαίσιο *εκτέλεση Python* 1.1 γίνεται χρήση της κλάσης `Point2` στο διερμηνέα της Python. Το σημείο p_2 αρχικοποιείται με όρισμα το σημείο p_1 ενώ το σημείο p_3 αρχικοποιείται από την πλειάδα των συντεταγμένων του σημείου p_2 (σημειώστε εδώ ότι η ιδιότητα `coordinates` ενός στιγμιότυπου της `Point2` είναι η πλειάδα

Κλάση Python 1.1: *Αναπαράσταση σημείων δύο διαστάσεων*

```
1  class Point2(object):
2    def __init__(self, x, y):
3      self.x, self.y = x, y
4    def __repr__(self):
5      return "Point2(%s, %s)" % (self.x, self.y)
6    @classmethod
7    def from_point2(cls, point2):
8      return cls(point2.x, point2.y)
9    @classmethod
10   def from_tuple(cls, tup):
11     return cls(tup[0], tup[1])
12   @property
13   def coordinates(self):
14     return self.x, self.y
15   @coordinates.setter
16   def coordinates(self, tup):
17     self.x, self.y = tup[0], tup[1]
```

Εκτέλεση Python 1.1: *Χρήση της κλάσης* `Point2` *στο διερμηνέα της Python. Παρατηρήστε τη χρήση των μεθόδων της κλάσης για την αρχικοποίηση με χρήση άλλου σημείου ή με τη χρήση πλειάδας.*

```
>>> from point import Point2
>>> p1 = Point2(3.5, -4)
>>> p2 = Point2.fromPoint2(p1)
>>> p3 = Point2.fromTuple(p2.coordinates)
>>> print p1, p2, p3
Point2(3.5, -4) Point2(3.5, -4) Point2(3.5, -4)
```

Κλάση Python 1.2: *Αναπαράσταση ευθυγράμμων τμημάτων στο επίπεδο*

```
1  class Segment2(object):
2      def __init__(self, start, end):
3          self.start = start
4          self.end = end
5      def __repr__(self):
6          return "Segment2(%s, %s)" % (self.start, self.end)
7      @classmethod
8      def from_segment2(cls, segment2):
9          return cls(segment2.start, segment2.end)
```

Εκτέλεση Python 1.2: *Χρήση της κλάσης* `Segment2` *στο διερμηνέα της Python*

```
>>> from point import Point2
>>> from segment import Segment2
>>> p = Point2(1.2, 3.21)
>>> q = Point2(-2, 2.34)
>>> s = Segment2(p, q)
>>> print s
Segment2(Point2(1.2, 3.21), Point2(-2.0, 2.34))
```

των συντεταγμένων του). Τελικά έχουμε τρία στιγμιότυπα σημείων με ακριβώς τις ίδιες συντεταγμένες.

1.1.2 Αναπαράσταση ευθυγράμμων τμημάτων

Μια κλάση της Python που τα στιγμιότυπά τους αναπαριστούν ευθύγραμμα τμήματα στο επίπεδο είναι η κλάση που φαίνεται στο πλαίσιο *κλάση Python* 1.2. Τα στιγμιότυπα της κλάσης έχουν δύο χαρακτηριστικά (attributes), τα άκρα του ευθύγραμμου τμήματος `start` και `end`.

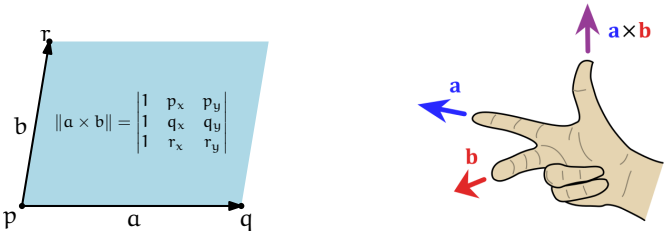
Στο πλαίσιο *εκτέλεση Python* 1.2 γίνεται χρήση της κλάσης `Segmenet2` στο διερμηνέα της Python. Τα p και q είναι δύο στιγμιότυπα της κλάσης `Point2` και χρησιμοποιούνται σαν ορίσματα στην αρχικοποίηση του s. Το s αναπαριστά το ευθύγραμμο τμήμα που αρχίζει από το σημείο p και τελειώνει στο σημείο q.

1.1.3 Εμβαδόν τριγώνου

Από το σχολείο γνωρίζουμε ότι το εμβαδόν ενός τριγώνου ισούται με το μισό του γινομένου του μήκους της βάσης του επί το μήκος του ύψους του. Όμως αυτός ο τύπος υπολογισμού του εμβαδού δεν είναι άμεσα χρήσιμος όταν πρέπει να υπολογιστεί το εμβαδόν ενός τριγώνου T με κορυφές τρία τυχαία σημεία $p = (p_x, p_y)$, $q = (q_x, q_y)$ και $r = (r_x, r_y)$. Ενώ ο υπολογισμός του μήκους της βάσης υπολογίζεται άμεσα σαν $\|a-b\|$, ο υπολογισμός του ύψους δεν είναι άμεσα διαθέσιμος από τις συντεταγμένες, εκτός αν το τρίγωνο είναι έτσι προσανατολισμένο ώστε κάποια πλευρά του να είναι παράλληλη με ένα από τους άξονες συντεταγμένων.

Εξωτερικό γινόμενο

Από τη γραμμική άλγεβρα γνωρίζουμε ότι το μέτρο του εξωτερικού γινομένου δύο διανυσμάτων είναι το εμβαδόν του παραλληλογράμμου που ορίζουν: αν a και b είναι δύο διανύσματα τότε $\|a \times b\|$ είναι το εμβαδό του παραλληλογράμμου με πλευρές τα a και b όπως φαίνεται και στο Σχήμα 1.1. Κάθε τρίγωνο μπορεί να θεωρηθεί σαν το μισό ενός παραλληλογράμμου κι έτσι έχουμε ένα άμεσο τρόπο για να υπολογίσουμε το εμβαδόν από τις συντεταγμένες: θέτουμε $a = q - p$ και $b = r - p$ και τότε το εμβαδόν του τριγώνου είναι το μισό της ποσότητας $a \times b$. Υπενθυμίζουμε πως οι συντεταγμένες του εξωτερικού γινομένου $a \times b$, με τα a



Σχήμα 1.1: *Εμβαδό παραλληλογράμμου και ο κανόνας του δεξιού χεριού.*

Συνάρτηση Python 1.1: *Υπολογίζει το διπλάσιο του εμβαδού του τριγώνου με κορυφές τα σημεία* p, q, r. *Ο προσανατολισμός των σημείων είναι ταυτόσημος με την τιμή που επιστρέφει η συνάρτηση.*

```
1 def area2(p, q, r):
2   return (r.y-p.y) * (q.x-p.x) - (q.y-p.y) * (r.x-p.x)

3 orientation = area2
```

και b θεωρούμενα ως διανύσματα στον Ευκλείδιο χώρο $\mathbb{R}^3$, δίνονται από τις 2×2 υποορίζουσες του παρακάτω πίνακα, ως προς την τελευταία γραμμή:

$$\begin{bmatrix} q_x - p_x & q_y - p_y & 0 \\ r_x - p_x & r_y - p_y & 0 \\ \Pi_x & \Pi_y & \Pi_z \end{bmatrix}$$

Το $a \times b$ είναι κάθετο στο επίπεδο των a, b και η τρίτη του συντεταγμένη δίνεται από την ορίζουσα:

$$\det \begin{bmatrix} q_x - p_x & q_y - p_y \\ r_x - p_x & r_y - p_y \end{bmatrix} = \det \begin{bmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{bmatrix}$$

Η ορίζουσα μηδενίζεται ανν τα τρία σημεία είναι συνευθειακά.

Στο πλαίσιο *συνάρτηση Python* 1.1 φαίνεται μια συνάρτηση Python που υπολογίζει το διπλάσιο του εμβαδού του τριγώνου με κορυφές τα σημεία p, q, r. Πρόκειται για τον υπολογισμό της τιμής της παραπάνω ορίζουσας. Στο πλαίσιο *εκτέλεση Python* 1.3 χρησιμοποιούμε τη συνάρτηση `area2` στο διερμηνέα της Python. Παρατηρούμε ότι οι τιμές του εμβαδού αλλάζουν ανάλογα με τη σειρά που περνάνε τα σημεία σαν ορίσματα στη συνάρτηση. Παρατηρήστε ότι το εμβαδόν του τριγώνου pqr είναι ένας θετικός αριθμός ενώ το εμβαδόν του τριγώνου qpr είναι ο αντίθετος αριθμός, δηλαδή ένας αρνητικός αριθμός. Συνδυάστε τώρα με αυτή την παρατήρηση τον κανόνα του δεξιού χεριού που φαίνεται στο Σχήμα 1.1: το εμβαδόν του τριγώνου pqr έχει θετικό πρόσημο γιατί η φορά του εξωτερικού γινομένου $(q-p) \times (r-p)$ έχει τρίτη συντεταγμένη με θετικό πρόσημο ενώ στην περίπτωση του τριγώνου qpr η τρίτη συντεταγμένη έχει αρνητικό πρόσημο.

Εκτέλεση Python 1.3: *Χρήση της συνάρτησης* `area2` *στον διερμηνέα της Python. Παρατηρήστε τις αρνητικές τιμές που δηλώνουν τον προσανατολισμό της επιφάνειας του τριγώνου σύμφωνα με τον κανόνα του δεξιού χεριού.*

```
>>> from point import Point2
>>> from utilities import area2
>>> p = Point(0, 0)
>>> q = Point(3, 0)
>>> r = Point(0, 4)
>>> area2(p, q, r)
12.0
>>> area2(q, p, r)
-12.0
>>> area2(p, q, q)
0.0
>>>
```

Κατηγόρημα Python 1.1: *Επιστρέφει* `True` *αν τα* p, q, r, *με αυτή ακριβώς τη σειρά, ορίζουν μια στροφή αντίθετη με τη φορά των δεικτών του ρολογιού.*

```
1 def ccw(p, q, r):
2     return area2(p, q, r) > 0
```

Το πρόσημο του εμβαδού του τριγώνου pqr μπορεί ισοδύναμα να εκφράσει τη φορά της στροφής που ορίζουν τα σημεία p, q και r με αυτή ακριβώς τη σειρά. Θεωρούμε δηλαδή ότι βαδίζουμε πάνω στο όριο του τριγώνου ξεκινώντας από το σημείο p, προς το σημείο q και θέλουμε να διαπιστώσουμε τη φορά της στροφής από το σημείο q προς το σημείο r. Αν η στροφή είναι προς τα αριστερά, δηλαδή αντίθετα με τη φορά κίνησης των δεικτών του ρολογιού (counter clockwise, ccw), τότε το πρόσημο του εμβαδού είναι θετικό. Στην αντίθετη περίπτωση η στροφή είναι σύμφωνη με τη φορά κίνησης των δεικτών του ρολογιού (clockwise, cw) και το πρόσημο του εμβαδού είναι αρνητικό. Το εμβαδόν του τριγώνου μπορεί να πάρει και μια τρίτη, ειδική τιμή, στην περίπτωση που τα τρία σημεία είναι συνευθειακά και η στροφή δεν ορίζεται. Με άλλα λόγια, ο υπολογισμός του εμβαδού του τριγώνου pqr είναι η ένδειξη για το ημιεπίπεδο της ευθείας pq που ανήκει το σημείο r.

1.1.4 Κατηγορήματα προσανατολισμού

Κατηγόρημα (predicate) καλείται ο έλεγχος μιας ιδιότητας ως προς ορισμένα γεωμετρικά δεδομένα. Ένα πιο θεμελιώδες κατηγόρημα είναι η σύγκριση πραγμα-

Κατηγόρημα Python 1.2: *Επιστρέφει* `True` *αν τα* p, q, r, *με αυτή ακριβώς τη σειρά, ορίζουν μια στροφή αντίθετη με τη φορά των δεικτών του ρολογιού ή αν το* r *είναι συνευθειακό με τα* p *και* q.

```
1 def ccwon(p, q, r):
2     return area2(p, q, r) >= 0
```

Κατηγόρημα Python 1.3: *Επιστρέφει* `True` *αν τα* p, q, r, *με αυτή ακριβώς τη σειρά, ορίζουν μια στροφή σύμφωνα με τη φορά των δεικτών του ρολογιού.*

```
1 def cw(p, q, r):
2     return area2(p, q, r) < 0
```

Κατηγόρημα Python 1.4: *Επιστρέφει* `True` *αν τα* p, q, r, *με αυτή ακριβώς τη σειρά, ορίζουν μια στροφή σύμφωνα με τη φορά των δεικτών του ρολογιού ή αν το* r *είναι συνευθειακό με τα* p *και* q.

```
1 def cwon(p, q, r):
2     return area2(p, q, r) <= 0
```

Κατηγόρημα Python 1.5: *Επιστρέφει* `True` *αν τα* p, q, r, *είναι συνευθειακά.*

```
1 def collinear(p, q, r):
2     return area2(p, q, r) == 0
```

Εκτέλεση Python 1.4: *Χρήση των κατηγορημάτων προσανατολισμού στο διερμηνέα της Python*

```
>>> from point import Point2
>>> from predicates import *
>>> p = Point2(-0.00342324, 2.03424345)
>>> q = Point2(23.47029054, 3.34344444)
>>> r = Point2(-2.1, -23.00009389)
>>> ccw(p, q, r)
False
>>> cw(p, q, r)
True
>>> collinear(p, q, r)
False
>>> collinear(p, q, q)
True
>>> collinear(q, q, q)
True
```

τικών αριθμών. Το κατηγόρημα αυτό είναι αρκετό για τους αλγορίθμους ταξινόμησης και συναντάται στην Υπολογσιτική Γεωμετρία σαν σύγκριση συντεταγμένων. Στη Λογική, ένα κατηγόρημα αποφασίζει την τιμή αλήθειας μιας πρότασης. Γενικά, η έξοδος του κατηγορήματος παίρνει, συνήθως δύο διακριτές τιμές. Είναι δυνατόν το κατηγόρημα να παίρνει και μια τρίτη τιμή, η οποία συνήθως αντιπροσωπεύει την εκφυλισμένη περίπτωση.

Στο πλαίσιο *κατηγόρημα Python* **??** υλοποιούμε στην Python τα βασικά κατηγορήματα προσανατολισμού τριών σημείων στο επίπεδο που βρίσκονται σε απόλυτη αντιστοιχία με το είδος των στροφών που περιγράψαμε στην προηγούμενη παράγραφο. Για να αποφασίσουμε την τιμή του κάθε κατηγορήματος χρησιμοποιούμε τη συνάρτηση `area2` και ελέγχουμε το πρόσημο της τιμής που επιστρέφει. Θετικό πρόσημο έχουμε όταν τα σημεία `p`, `q`, `r` ορίζουν μια στροφή αντίθετη με τη φορά των δεικτών του ρολογιού (συνάρτηση `ccw`), αρνητικό πρόσημο όταν η στροφή είναι σύμφωνη με τη φορά των δεικτών του ρολογιού (συνάρτηση `cw`), ενώ η τιμή της `area2` είναι 0 όταν το τρίγωνο είναι εκφυλισμένο σε ένα ευθύγραμμο τμήμα ή ένα σημείο (συνάρτηση `collinear`).

Στο πλαίσιο *εκτέλεση Python* 1.4 χρησιμοποιούμε στον διερμηνέα της Python τα διάφορα κατηγορήματα προσανατολισμού. Ορίζουμε τρία σημεία `p`, `q` και `r` και ελέγχουμε τις τιμές των κατηγορημάτων. Φαίνεται ότι η στροφή `pqr` είναι σύμφωνη με τη φορά κίνησης των δεικτών του ρολογιού ενώ στις περιπτώσεις που το τρίγωνο εκφυλίζεται σε ένα ευθύγραμμο τμήμα ή ένα σημείο, η τιμή της συνάρτησης `collinear` είναι `True`.

1.1.5 Τομή ευθυγράμμων τμημάτων

Στο σημείο αυτό έχουμε τα εφόδια για την υλοποίηση ενός πιο πολύπλοκου κστηγορήματος, αυτού της τομής δύο ευθύγραμμων τμημάτων. Θα χρησιμοποιήσουμε δηλαδή τα κατηγορήματα προσανατολισμού σαν ένα αποτελεσματικό εργαλείο για το πρόβλημα απόφασης: Με δεδομένα δύο ευθύγραμμα τμήματα $s_1 = (\mathtt{p}, \mathtt{q})$ και $s_2 = (\mathtt{r}, \mathtt{t})$ πως θα αποφασίσουμε αν τα δύο ευθύγραμμα τμήματα τέμνονται; Σημειώστε ότι ο ακριβής υπολογισμός του σημείου τομής είναι μια διαδικασία επιδεκτική λαθών (error prone operation) και τέτοιες διαδικασίες θέλουμε να τις αποφεύγουμε και να πληρώνουμε το υπολογιστικό κόστος μόνο όταν αυτό είναι απαραίτητο (όταν για παράδειγμα η είσοδος έχει ευθύγραμμα τμήματα με ελάχιστα σημεία τομής και το ζητούμενο είναι να υπολογίσουμε όλα τα σημεία τομής).

Το κατηγόρημα `intersect(s1, s2)` είναι αληθές αν και μόνο αν τα σημεία `p` και `q` δεν είναι μαζί "στην ίδια πλευρά" του s_2 και τα σημεία `r` και `t` δεν είναι μαζί "στην ίδια πλευρά" του s_1. Η υλοποίηση του κατηγορήματος είναι η εξής: τα σημεία `p` και `q` δεν είναι μαζί "στην ίδια πλευρά" του s_2 αν οι τριάδες `pqr` και `pqs` έχουν διαφορετικό προσανατολισμό, η μία θετικό ενώ η άλλη αρνητικό και ταυτόχρονα το ίδιο πρέπει να συμβαίνει και για τις τριάδες `rtp` και `rtq`.

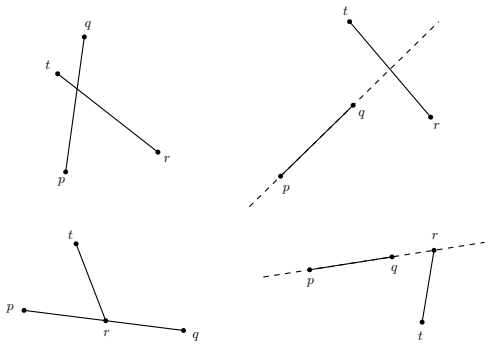
Αν τα s_1 και s_2 τέμνονται στο εσωτερικό τους τότε τα `r` και `t` χωρίζονται από την ευθεία που περνά από τα `p` και `q`: το `r` βρίσκεται στη μια μεριά και το `q` στην άλλη. Παρόμοια τα `p` και `q` χωρίζονται από την ευθεία που περνά από τα `r` και `t`

Κατηγόρημα Python 1.6: *Επιστρέφει* `True` *ανν τα ευθύγραμμα τμήματα* `segment1` *και* `segment2` *τέμνονται σε ένα σημείο αυστηρά στο εσωτερικό των τμημάτων.*

```
def intersectProperly(segment1, segment2):
    p, q = segment1.start, segment1.end
    r, t = segment2.start, segment2.end

    if collinear(p,q,r) or collinear(p,q,t) or collinear(r,t,p) or
        collinear(r,t,q):
        return False
    else
        return orientation(p,q,r) <> orientation(p,q,t) and
            orientation(r,t,p) <> orientation(r,t,q)
```

(δείτε το σχήμα 1.2). Καμιά από τις δύο συνθήκες δεν είναι από μόνη της ικανή για να εγγυηθεί την τομή, όπως φαίνεται στο σχήμα 1.2, όμως είναι σαφές πως όταν ισχύουν και οι δύο μαζί τότε αναγκαστικά τα ευθύγραμμα τμήματα τέμνονται. Η τελευταία παρατήρηση μας οδηγεί σε μια απλή κωδικοποίηση του κατηγορήματος `intersectProperly` που αποφασίζει αν δύο ευθύγραμμα τμήματα τέμνονται σε ένα σημείο που βρίσκεται εσωτερικά και στα δύο τμήματα αν γνωρίζουμε ότι δεν υπάρχουν τρία από τα τέσσερα άκρα που είναι συνευθειακά.



Σχήμα 1.2: *Τομές ευθυγράμμων τμημάτων.*

Στη συνέχεια πρέπει να αντιμετωπίσουμε την ειδική περίπτωση που τα ευθύγραμμα τμήματα δεν τέμνονται αυστηρά σε ένα σημείο στο εσωτερικό τους. Αυτό μπορεί να συμβεί αν το ένα άκρο ενός τμήματος, έστω το r βρίσκεται κάπου στο εσωτερικό του άλλου τμήματος pq (δείτε το σχήμα 1.2). Αυτό μπορεί να συμβεί μόνο αν τα σημεία p, q, r είναι συνευθειακά. Όμως η συνθήκη δεν είναι αναγκαία όπως μπορούμε να δούμε στην αντίστοιχη περίπτωση του σχήματος 1.2. Πρέπει να αποφασίσουμε αν το σημείο r βρίσκεται ανάμεσα στα p και q.

Για να κωδικοποιήσουμε στην Python το κατηγόρημα `between(p,q,r)` παρατηρούμε ότι αυτό θα είναι αληθές μόνο αν το σημείο r βρίσκεται πάνω στην ευθεία που περνά από τα p και q. Αν το pq δεν είναι κάθετο στον οριζόντιο άξονα, το r βρίσκεται πάνω στο pq ανν η x-συντεταγμένη του r βρίσκεται μέσα στο διάστημα που ορίζουν οι x-συντεταγμένες των p και q. Αν το pq είναι κάθετο στον οριζόντιο άξονα, το r βρίσκεται πάνω στο pq ανν η y-συντεταγμένη του r βρίσκεται μεα

Κατηγόρημα Python 1.7: *Επιστρέφει* `True` *ανν τα σημεία* p, q, r *είναι συνευθειακά και το* r *βρίσκεται πάνω στο κλειστό διάστημα* pq.

```
def between(p, q, r):
    if not collinear(p, q, r):
        return False
    if p.x != q.x:
        return p.x <= r.x <= q.x or p.x >= r.x >= q.x
    else:
        return p.y <= r.y <= q.y or p.y >= r.y >= q.y
```

Κατηγόρημα Python 1.8: *Επιστρέφει* `True` *ανν τα ευθύγραμμα τμήματα* `segment1` *και* `segment2` *τέμνονται αυστηρά ή αν το άκρο του ενός τμήματος βρίσκεται πάνω στο άλλο τμήμα και ανάμεσα στα άκρα του.*

```
def intersect(segment1, segment2):
    p, q = segment1.start, segment1.end
    r, t = segment2.start, segment2.end
    if intersectProperly(segment1, segment2):
        return True
    else
        return between(p,q,r) or between(p,q,t) or between(r,t,p) or
            between(r,t,q)
```

στο διάστημα που ορίζουν οι y-συντεταγμένες των p και q. Η κωδικοποίηση στην Python φαίνεται στο πλαίσιο *κατηγόρημα Python* 1.7.

Σε αυτό το σημείο έχουμε όλα τα εργαλεία που θα μας επιτρέψουν να υλοποιήσουμε τον έλεγχο για την τομή δύο ευθύγραμμων τμημάτων. Δύο ευθύγραμμα τμήματα τέμνονται ανν τέμνονται αυστηρά ή το ένα άκρο του ενός τμήματος βρίσκεται μεταξύ των δύο άκρων του άλλου τμήματος. Ο έλεγχος για τη δεύτερη συνθήκη γίνεται με τέσσερις εφαρμογές του κατηγορήματος `between` (δείτε το πλαίσιο *κατηγόρημα Python* 1.8).

1.1.6 Τριγωνοποίηση Απλού Πολυγώνου

Στο σημείο αυτό είμαστε εφοδιασμένοι με όλα τα απαραίτητα προγραμματιστικά εργαλεία για να υλοποιήσουμε τον πρώτο μας γεωμετρικό αλγόριθμο. Το πρόβλημα που θα αντιμετωπίσουμε είναι αυτό της τριγωνοποίησης ενός απλού πολυγώνου. Τα απλά πολύγωνα είναι από τα βασικότερα σχήματα στην Υπολογιστική Γεωμετρία καθώς είναι το μοντέλο που αναπαριστά διάφορες πρακτικές εφαρμογές που ποικίλουν από τη φύλαξη μουσείων και την οπτική επικοινωνία σε γενικότερα περιβάλλοντα με πολυγωνικά εμπόδια έως στην κάλυψη περιοχών από δίκτυα ασύρματων επικοινωνιών. Κατά την επίλυση γενικών προβλημάτων είναι μια συνηθισμένη τακτική να υποδιαιρούμε ένα σχήμα σε απλούστερα έτσι ώστε το πρόβλημα να αντιμετωπίζεται ευκολότερα. Ένα απλό πολύγωνο θα μπορούσε να υποδιαιρεθεί σε οποιοδήποτε σχήμα και το απλούστερα από αυτά είναι το τρίγωνο.

Ορισμός 2. *Μια ακολουθία ακμών* $e_1, \ldots, e_n$ *ορίζει ένα* απλό πολύγωνο ανν δύο ακμές του πολυγώνου που δεν είναι γειτονικές δεν τέμνονται και οι ακμές τέμνονται

Κλάση Python 1.3: *Περιγράφει ένα πολύγωνο σαν μια λίστα κορυφών και μια λίστα ακμών. Η κλάση αρχικοποιείται είτε από μια λίστα σημείων, στιγμιοτύπων της κλάσης* `Point2`*, είτε από μια λίστα πλειάδων.*

```
class Polygon2(object):
    def __init__(self, vertices=[]):
        self.__edges = []
        if vertices:
            self.vertices = vertices

    def __getitem__(self, index):
        return self.__vertices[index % len(self)]

    def __len__(self):
        return len(self.__vertices)

    def index(self, item):
        return self.__vertices.index(item)

    @property
    def vertices(self):
        for vertex in self.__vertices:
            yield vertex

    @vertices.setter
    def vertices(self, vertices):
        self.__vertices = [Point2.from_point2(x) for x in vertices]
        self.__edges = []
        lastvertex = None
        currentvertex = self.__vertices[0]
        for vertex in self.__vertices[1:]:
            edge = Segment2(currentvertex, vertex)
            self.__edges.append(edge)
            currentvertex = vertex
            lastvertex = vertex
        if lastvertex: # Close the polygon boundary
            self.__edges.append(Segment2(vertex, self.__vertices[0]))

    @property
    def edges(self):
        for edge in self.__edges:
            yield edge
```

μόνο στις κορυφές τους:

- $e_i \cap e_j = \emptyset$ αν $|i - j| > 1$
- $e_n \cap e_1$ και κάθε $e_i \cap e_{i+1}$ είναι κορυφές.

Στο σχήμα XX φαίνονται δύο απλά πολύγωνα (αριστερά) και δύο μη απλά πολύγωνα (δεξιά). Στη συνέχεια με τον όρο *πολύγωνο* θα εννούμε *απλό πολύγωνο*.

Αναπαράσταση πολυγώνων στην Python

Μια κλάση που περιγράφει πολύγωνα στην Python φαίνεται στο πλαίσιο XX. Η κλάση `Polygon2` έχει δύο ιδιότητες, τις `vertices` και `edges` που είναι αντίστοιχα οι λίστες των κορυφών και των πλευρών του πολυγώνου. Το πολύγωνο δίνεται συνήθως στην είσοδο με μια συγκεκριμένη σειρά των κορυφών της περιμέτρου του, για παράδειγμα με τη σειρά που θα τις δώσει με το ποντίκι ο χρήστης όταν ορίζει το πολύγωνο. Ορίζεται έτσι μια σειρά εισαγωγής των κορυφών κατά την εισαγωγή της λίστας των κορυφών του πολυγώνου. Προς το παρόν ας θεωρήσουμε πως ο χρήστης εισάγει με κάποια σειρά τη λίστα κορυφών ενός απλού πολυγώνου.

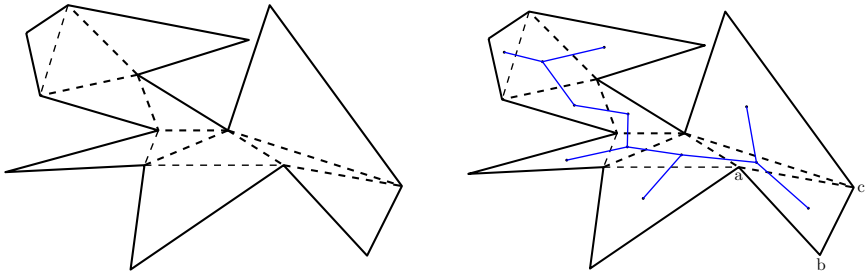
Η διαδικασία ανάθεσης στην ιδιότητα `vertices` της κλάσης `Polygon2` αρχικοποιεί τα ιδιωτικά χαρακτηριστικά `__vertices` και `__edges` της κλάσης. Παρατηρήστε πως τα ιδιώματα της Python μας επιτρέπουν να κωδικοποιήσουμε ξεκάθαρα τη δημιουργία της λίστας των πλευρών του πολυγώνου και να έχουμε πρόσβαση με δείκτες σε κάθε κορυφή του. Με δεδομένο ένα στιγμιότυπο της κλάσης `Polygon2` η αναφορά στις ιδιότητες `vertices` και `edges` ενεργοποιεί τους αντίστοιχους γεννήτορες (generators) επιτρέποντας έτσι τη διάσχιση των κορυφών και των ακμών του πολυγώνου.

Διαγώνιος ενός πολυγώνου P είναι ένα ευθύγραμμο τμήμα pq που βρίσκεται μέσα στο πολύγωνο και το τέμνει σε δύο διαφορετικές κορυφές του. Με άλλα λόγια, η τομή της περιμέτρου ∂P με τη διαγώνιο pq είναι ακριβώς τα άκρα p και q, δύο διαφορετικές κορυφές του P.

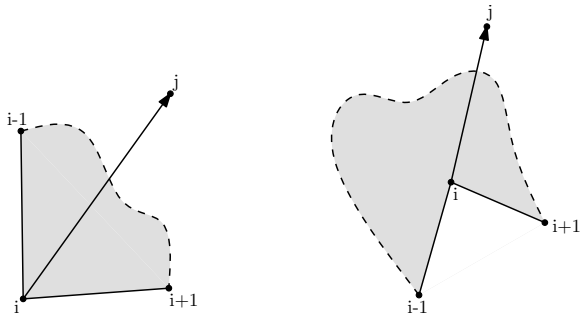
Ορισμός 3. *Τριγωνοποίηση ενός πολυγώνου είναι το μεγαλύτερο σύνολο από διαγώνιους του πολυγώνου που είτε δεν τέμνονται είτε διαθέτουν μια κοινή κορυφή*

Ο δυϊκός γράφος T μιας τριγωνοποίησης ενός απλού πολυγώνου έχει ένα κόμβο για κάθε τρίγωνο και ακμές μεταξύ των κόμβων ανν τα αντίστοιχα τρίγωνα μοιράζονται μια διαγώνιο. Αποδεικύεται εύκολα ότι ο γράφος T είναι δέντρο με κόμβους βαθμού το πολύ 3. Οι κόμβοι με βαθμό 1 είναι τα φύλλα του T, κόμβοι με βαθμό 2 βρίσκονται πάνω σε μονοπάτια του δέντρου ενώ κόμβοι με βαθμό 3 είναι σημεία διακλάδωσης των μονοπατιών. Παρατηρήστε ότι ο T είναι ένα δυαδικό δέντρο αν θωρήσουμε σαν ρίζα οποιοδήποτε κόμβο βαθμού 1 ή 2.

Λέμε ότι τρείς συνεχόμενες κορυφές ενός πολυγώνου, a, b, c σχηματίζουν ένα *αυτί* αν το ac είναι διαγώνιος του πολυγώνου. Δύο αυτιά είναι *μη επικαλυπτόμενα* αν τα εσωτερικά των τριγώνων τους είναι ξένα μεταξύ τους. Κάθε πολύγωνο με



Σχήμα 3.1: *Στα αριστερά μια τριγωνοποίηση ενός απλού πολυγώνου. Στα δεξιά με μπλέ χρώμα φαίνεται ο δυϊκός γράφος της τριγωνοποίησης. Το τρίγωνο* abc *είναι ένα "αυτί" του παραλληλογράμμου.*



Σχήμα 3.2: *Η διαγώνιος* ij *είναι εσωτερικά του κώνου που ορίζουν οι κορυφές* v_{i-1}, v_i, v_{i+1} *στην περίπτωση κυρτής κορυφής στα αριστερά και μη κυρτής στα δεξιά.*

Κατηγόρημα Python 3.1: *Επιστρέφει* `True` *αν το ευθύγραμμο τμήμα* `segment` *είναι εσωτερική ή εξωτερική διαγώνιος του πολυγώνου* `polygon`.

```
def is_internal_or_exterrnal_diagonal(segment, polygon):
    p, q = polygon.index(segment.start), polygon.index(segment.end)
    for edge in polygon.edges:
        r, t = polygon.index(edge.start), polygon.index(edge.end)
        # skip edges incident to segment
        if not ((p==r) or (p==t) or (q==r) or (q==t))
            if intersect(segment, edge):
                return False
    return True
```

$n \geq 4$ κορυφές έχει τουλάχιστο δύο μη επικαλυπτόμενα αυτιά αφού ένα δυαδικό δέντρο με δύο ή παραπάνω κόμβους πρέπει να έχει τουλάχιστο δύο φύλλα.

Ένας απλοϊκός αλγόριθμος κατασκευάζει μια τριγωνοποίηση ενός απλού πολυγώνου με n κορυφές εξετάζοντας $O(n)$ πιθανές "διαγωνίους αυτιών" $(i, i+2)$, για $i = 0, \ldots, n-1$. Όταν ο αλγόριθμος εντοπίσει ένα αυτί τότε αναφέρει στην έξοδο τη διαγώνιο $(i, i+2)$, αφαιρεί το αυτί και συνεχίζει τη διαδικασία για το υπόλοιπο πολύγωνο.

Από τον ορισμό τους οι διαγώνιοι ενός απλού πολυγώνου χαρακτηρίζονται από δύο ιδιότητες: δεν τέμνουν τις πλευρές του πολυγώνου και βρίσκονται ολόκληρες στο εσωτερικό του. Αν αγνοήσουμε τη διάκριση μεταξύ εσωτερικών και εξωτερικών διαγωνίων τότε η εύρεση διαγωνίων είναι σχετικά εύκολη: για κάθε πλευρά e του πολυγώνου που δεν προσπίπτει στα άκρα μιας πιθανής διαγωνίου s, διαπιστώνουμε αν η e τέμνει την s. Αν δεν υπάρχει πλευρά που τέμνει την s, τότε η s είναι διαγώνιος. Στο πλαίσιο *κατηγόρημα Python* 3.1 φαίνεται ένα κατηγόρημα Python που επιστέφει `True` ανν ένα ευθύγραμμο τμήμα είναι εσωτερική ή εξωτερική διαγώνιος ενός πολυγώνου.

Στην προηγούμενη συνάρτηση διασφαλίσαμε ότι η διαγώνιος s δεν τέμνει καμία πλευρά του πολυγώνου. Η διάκριση των εσωτερικών διαγωνίων γίνεται τοπικά σε ένα άκρο της s. Αν $s = (i, j)$ τότε η s είναι εσωτερική διαγώνιος αν βρίσκεται εντός του "κώνου" που ορίζουν οι πλευρές $(i-1, i)$ και $(i, i+1)$: Σημειώστε ότι αφού η (i, j) δεν τέμνει καμία πλευρά του πολυγώνου τότε ο έλεγχος για το αν πρόκειται για εσωτερική διαγώνιο εξαρτάται αποκλειστικά από τις προσπίπτουσες πλευρές στο άκρο i (ομοίως και για το j), συνεπώς ο έλεγχος μπορεί (πρέπει) να έχει σταθερή χρονική πολυπλοκότητα.

Στο πλαίσιο *κατηγόρημα Python* 3.2 φαίνεται το κατηγόρημα που επιστρέφει `True` ανν η διαγώνιος `diagonal` είναι εσωτερική του πολυγώνου `polygon`. Όπως φαίνεται και στο σχήμα 3.2 πρέπει να διακρίνουμε δύο περιπτώσεις για την κορυφή που αντιστοιχεί στην αρχή της διαγωνίου, την κυρτή και την μη κυρτή κορυφή. Παρατηρήστε στο σχήμα 3.2 ότι και στις δύο περιπτώσεις για να ανήκει η διαγώνιος στον "κώνο" που σχηματίζουν οι προσπίπτουσες ακμές του πολυγώνου στην αρχή της διαγωνίου πρέπει να ισχύει ότι η κορυφή $\texttt{vim1} = v_{i-1}$ είναι αριστερά της $(\texttt{vi}, \texttt{vj}) = v_i v_j$, δηλαδή να ισχύει `ccw(vi, vj, vim1)`. Ταυτόχρονα η κορυφή $\texttt{vip1} = v_{i+1}$ είναι επίσης αριστερά της $(\texttt{vj}, \texttt{vi}) = v_j v_i$, πρέπει δηλαδή να ισχύει

Κατηγόρημα Python 3.2: *Επιστρέφει* `True` *αν η διαγώνιος* `diagonal` *είναι εσωτερική διαγώνιος του πολυγώνου* `polygon`*.*

```
def is_internal(diagonal, polygon):
    ...
    ...
    ...
    if ...:
        return ...
    else:
        return ...
```

Αλγόριθμος Python 3.1: *Παίρνει στην είσοδο ένα απλό πολύγωνο και επιστρέφει στην έξοδο τη λίστα των διαγωνίων που τριγωνοποιούν το πολύγωνο σύμφωνα με την μέθοδο της "κοπής αυτιών".*

```
def triangulate(polygon):
    diagonals = []
    while len(polygon) > 3:
        for edge in polygon(edges):
            i = polygon.index(edge.start)
            if diagonal(polygon[i], polygon[i+1], polygon[i+2]):
                diags.append(Segment2(polygon[i], polygon[i+2]))
                del(polygon[i+1])
                break
    return diagonals
```

`ccw(vj,vi,vip1)`.

Τέλος, στο πλαίσιο *αλγόριθμος Python* 3.1 διατυπώνουμε τον πρώτο μας αλγόριθμο στην Python, που λαμβάνει στην είσοδο ένα πολύγωνο και δίνει στην έξοδο μια λίστα με τις διαγωνίους της τριγωνοποίησης που προκύπτει με την απλοϊκή μέθοδο της "κοπής αυτιών" που περιγράψαμε παραπάνω.