ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ
ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΠΟΛΙΤΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΔΟΜΟΣΤΑΤΙΚΗΣ

Ακαδ. Έτος: 2016-2017

Μάθημα: Εφαρμογές Η/Υ

Δευτέρα, 24/10/2016

Διδάσκοντες:
Ν.Δ. Λαγαρός (Επικ. Καθηγητής), Αθ. Στάμος (ΕΔΙΠ), Χ. Φραγκουδάκης (ΕΔΙΠ)

Παραδείγματα για την 4η παράδοση – Αντικειμενοστραφής προγραμματισμός

1. Υπολογισμός κέντρου βάρους και ροπής αδρανείας σύνθετης διατομής με αντικείμενα

Να συνταχθεί πρόγραμμα σε Python που να υπολογίζει τη ροπή αδρανείας σύνθετης διατομής που αποτελείται από στοιχειώδεις ορθογωνικές και τριγωνικές διατομές χρησιμοποιώντας αντικειμενοστραφή προγραμματισμό. Επίσης να τυπώνει το κ.β. τη ροπή αδρανείας και την ακτίνα αδρανείας σε κάθε επιμέρους διατομή και τη σύνθετη διατομή.

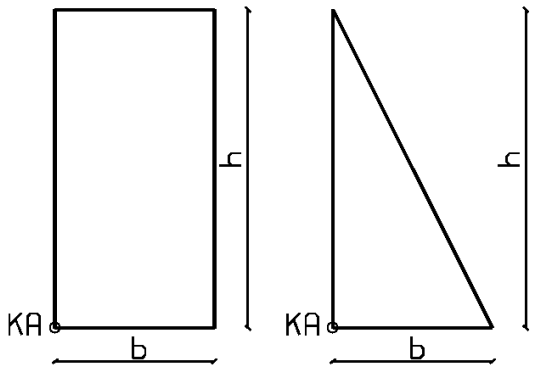
Το κ.β. και η ροπή αδρανείας της σύνθετης διατομής υπολογίζεται ως:

$$x_{c,ολ}=\frac{\sum_i x_{c,i}A_i}{\sum_i A_i} \quad , \quad y_{c,ολ}=\frac{\sum_i y_{c,i}A_i}{\sum_i A_i} \quad , \quad I_{x,ολ}=\sum_i \left[I_{x,i}+(y_{c,i}-y_{c,ολ})^2 A_i\right]$$

όπου I_x η ροπή αδρανείας, x_c,y_c οι συντεταγμένες του κ.β. και A το εμβαδόν μίας διατομής. Η ακτίνα αδρανείας i_x μίας διατομής υπολογίζεται ως:

$$i_x=\sqrt{\frac{I_x}{A}}$$

Το κέντρο βάρους, η ροπή αδρανείας και το εμβαδόν για ορθογωνική διατομή διαστάσεων b, h υπολογίζεται ως:

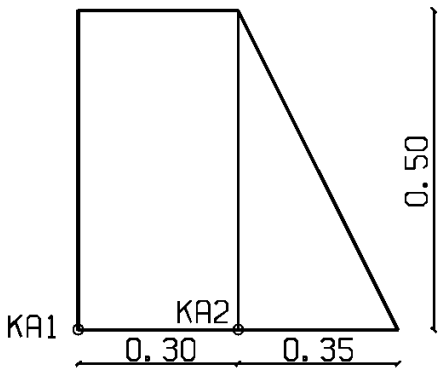


$$x_c=x_{KA}+\frac{b}{2} \quad , \quad y_c=y_{KA}+\frac{h}{2} \quad , \quad I_x=\frac{b\,h^3}{12} \quad , \quad A=b\,h$$

όπου x_{KA}, y_{KA} είναι οι συντεταγμένες τού κάτω αριστερά σημείου του ορθογωνίου. Για διατομή μορφής ορθογωνίου τριγώνου (με την κορυφή που αντιστοιχεί στην ορθή γωνία κάτω αριστερά) οι τύποι γίνονται:

$$x_c=x_{KA}+\frac{b}{3} \quad , \quad y_c=y_{KA}+\frac{h}{3} \quad , \quad I_y=\frac{b\,h^3}{36} \quad , \quad A=\frac{b\,h}{2}$$

Χρησιμοποιείστε το πρόγραμμα για να βρεθεί η ροπή αδρανείας της παρακάτω σύνθετης διατομής:



Λύση 1

Σε κάθε διατομή υπολογίζεται το κ.β. και η ροπή αδρανείας με διαφορετικό τρόπο (διαφορετικούς τύπους). Αυτό σημαίνει ότι χρειάζεται μία κλάση αντικειμένων για κάθε διατομή. Για να μπορέσουμε να χρησιμοποιήσουμε τα αντικείμενα που θα προέλθουν από τις κλάσεις πιο εύκολα (χωρίς εντολές διακλάδωσης), οι μέθοδοι που υπολογίζουν το κ.β., τη ροπή αδρανείας και άλλα στοιχεία της διατομής θα έχουν το ίδιο όνομα σε κάθε κλάση.

```
# -*- coding: iso-8859-7 -*-
from math import sqrt
"Υπολογισμός κ.β. και ροπής αδρανείας σύνθετης διατομής."

class Orth:
    "Ορθογωνική διατομή για υπολογισμό ροπής αδρανείας."
    def __init__(self, b, h, xka, yka):
        "Καθορισμός των γεωμετρικών στοιχείων της διατομής."
        self.b = b
        self.h = h
        self.xka = xka
        self.yka = yka

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
        return self.xka + self.b/2, self.yka + self.h/2

    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.b *self.h**3 / 12

    def ypolEmb(self):
        "Υπολογισμός εμβαδού διατομής."
        return self.b * self.h

    def ypolAktina(self):
        "Υπολογισμός ακτίνας αδρανείας διατομής."
        Ix = self.ypolAdran()
        A = self.ypolEmb()
        return sqrt(Ix/A)

class Trig:
    "Τριγωνική διατομή για υπολογισμό ροπής αδρανείας."
    def __init__(self, b, h, xka, yka):
        "Καθορισμός των γεωμετρικών στοιχείων της διατομής."
        self.b = b
        self.h = h
        self.xka = xka
        self.yka = yka

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
```

```
        return self.xka + self.b/3, self.yka + self.h/3

    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.b *self.h**3 / 36

    def ypolEmb(self):
        "Υπολογισμός εμβαδού διατομής."
        return self.b * self.h / 2

    def ypolAktina(self):
        "Υπολογισμός ακτίνας αδρανείας διατομής."
        Ix = self.ypolAdran()
        A = self.ypolEmb()
        return sqrt(Ix/A)

def synkb(diats):
    "Υπολογισμός κ.β. σύνθετης διατομής."
    sx = sy = sA = 0
    for diat in diats:
        A = diat.ypolEmb()
        xc, yc = diat.ypolKb()
        sx += yc*A
        sy += xc*A
        sA += A
    return sy/sA, sx/sA

def synRopadr(diats, synxc, synyc):
    "Υπολογισμός ροπής αδρανείας σύνθετης διατομής."
    synIx = 0
    for diat in diats:
        A = diat.ypolEmb()
        xc, yc = diat.ypolKb()
        Ix = diat.ypolAdran()
        synIx += Ix + A * (yc-synyc)**2
    return synIx

def pyMain():
    "Standalone main program: example of inertia moments."
    diats = (  Orth(0.30, 0.50, 0.00, 0.00),
               Trig(0.35, 0.50, 0.30, 0.00),
            )
    xcs, ycs = synkb(diats)
    Ixs = synRopadr(diats, xcs, ycs)
    for diat in diats:
        xc, yc = diat.ypolKb()
        Ix = diat.ypolAdran()
        ix = diat.ypolAktina()
        print(xc, yc, Ix, ix)
    print("------------------------------------------------------------------------
-------------")
    As = sum([diat.ypolEmb() for diat in diats])
    ixs = sqrt(Ixs/As)
    print(xcs, ycs, Ixs, ixs)

pyMain()
```

Στη συνάρτηση `pyMain()` η κλήση της κλάσης `Orth()` και της κλάσης `Trig()` δημιουργεί στιγμιότυπα (instances) αυτών των κλάσεων τα οποία αποθηκεύονται σε μία πλειάδα. Δηλαδή τα στοιχεία της πλειάδας diats είναι στιγμιότυπα (διαφορετικών) κλάσεων. Παρεμπιπτόντως η πλειάδα είναι επίσης μία κλάση στην Python, η οποία ονομάζεται tuple. Για παράδειγμα η εντολή `a=tuple()` ισοδυναμεί με την εντολή `a=()`. Υπό αυτή την έννοια η μεταβλητή diats είναι και αυτή ένα στιγμιότυπο (instance), και συγκεκριμένα ένα στιγμιότυπο της κλάσης tuple. Συνηθίζεται τα ονόματα κλάσεων να αρχίζουν από κεφαλαίο γράμμα, ενώ τα

στιγμιότυπα των κλάσεων (class instances) και οι κοινές μεταβλητές να αρχίζουν από πεζό γράμμα (εξαιρούνται οι κλάσεις που δίνονται έτοιμες από την Python όπως η κλάση tuple).
Παρατηρούμε ότι στον κώδικα για τον υπολογισμό της σύνθετης διατομής δεν υπάρχουν εντολές διακλάδωσης (if), πράγμα που βελτιώνει κατά πολύ την αναγνωσιμότητα του κώδικα, και μικραίνει σημαντικά την πιθανότητα σφάλματος (bug). Ο κώδικας αποτελείται από μία επανάληψη στην οποια δεν υπάρχουν if. Αυτός είναι ο απλούστερος δυνατός (για μη τετριμμένα προβλήματα) και είναι συνηθισμένος στο αντικειμενοστραφή προγραμματισμό. Επιπλέον η συνάρτηση που υπολογίζει τη σύνθετη διατομή δεν γνωρίζει πώς υπολογίζεται το κ.β., τη ροπή αδρανείας και το εμβαδόν μίας διατομής, αυτό είναι αρμοδιότητα της αντίστοιχης κλάσης. Έτσι η σύνταξη των κλάσεων και της συνάρτησης υπολογισμού σύνθετης διατομής είναι τελείως ανεξάρτητες, μπορούν αποσφαλματωθούν ανεξάρτητα ή και να συνταχθούν ανεξάρτητα ακόμα και από διαφορετικά άτομα. Αυτό μειώνει ακόμα περισσότερο τις πιθανότητες να παραμείνει κάποιο σφάλμα στον κώδικα και μειώνει το χρόνο ανάπτυξης ενός προγράμματος.

Λύση 2 με κληρονομικότητα

Αν πολλές ομοειδείς κλάσεις έχουν κοινές κάποιες μεθόδους, είναι χρήσιμο να βάζουμε τις κοινές μεθόδους σε μία βασική (base class ή super class) για να αποφύγουμε επικάλυψη του κώδικα (code duplication – ο ίδιος κώδικας σε δύο η περισσότερα σημεία του προγράμματος), η οποία είναι και επίπονη και επιρρεπής σε λάθη. Στη συνέχεια οι ομοειδείς κλάσεις γίνονται παράγωγες κλάσεις (derived classes) της βασικής κλάσης και “κληρονομούν” (inherit) τις μεθόδους της βασικής κλάσης.
Εδώ παρατηρούμε ότι οι μέθοδοι __init__() και ypolAktina() είναι κοινές στις κλάσεις Orth() και Trig() και έτσι θα μεταφερθούν σε μια βασική Diatomh(). Οι κλάσεις Orth() και Trig() θα γίνουν παράγωγες κλάσεις της βασικής κλάσης Diatomh και θα κληρονομήσουν (inherit) τις μεθόδους της.

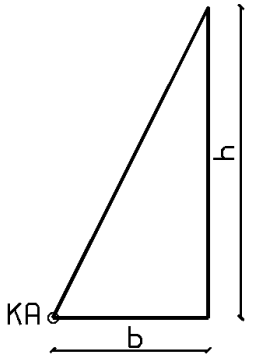
```
# -*- coding: iso-8859-7 -*-
from math import sqrt
"Υπολογισμός κ.β. και ροπής αδρανείας σύνθετης διατομής."

class Diatomh:
    "Βασική διατομή για υπολογισμό ροπής αδρανείας."
    def __init__(self, b, h, xka, yka):
        "Καθορισμός των γεωμετρικών στοιχείων της διατομής."
        self.b = b
        self.h = h
        self.xka = xka
        self.yka = yka

    def ypolAktina(self):
        "Υπολογισμός ακτίνας αδρανείας διατομής."
        Ix = self.ypolAdran()
        A = self.ypolEmb()
        return sqrt(Ix/A)

class Orth(Diatomh):
    "Ορθογωνική διατομή για υπολογισμό ροπής αδρανείας."

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
        return self.xka + self.b/2, self.yka + self.h/2

    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.b *self.h**3 / 12

    def ypolEmb(self):
        "Υπολογισμός εμβαδού διατομής."
        return self.b * self.h

class Trig(Diatomh):
    "Τριγωνική διατομή για υπολογισμό ροπής αδρανείας."

    def ypolKb(self):
```

```
        "Υπολογισμός κ.β. διατομής."
        return self.xka + self.b/3, self.yka + self.h/3

    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.b *self.h**3 / 36

    def ypolEmb(self):
        "Υπολογισμός εμβαδού διατομής."
        return self.b * self.h / 2
```

Οι συναρτήσεις synkb(), synRopadr και pyMain() παραμένουν οι ίδιες.

2. Υπολογισμός κέντρου βάρους και ροπής αδρανείας σύνθετης διατομής με αντικείμενα: προσθήκη νέας στοιχειώδους διατομής

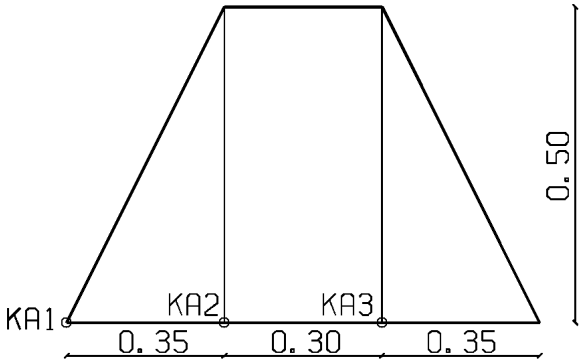
Στο προηγούμενο παράδειγμα, να προστεθεί ακόμα μία στοιχειώδης διατομή, η τριγωνική διατομή του σχήματος:



Το κέντρο βάρους, η ροπή αδρανείας και το εμβαδόν της διατομής (διαστάσεων b, h) υπολογίζεται ως:

$$x_c = x_{KA} + \frac{b}{3} \quad , \quad y_c = y_{KA} + \frac{2h}{3} \quad , \quad I_y = \frac{b h^3}{36} \quad , \quad A = \frac{b h}{2}$$

όπου x_{KA}, y_{KA} είναι οι συντεταγμένες τού κάτω αριστερά σημείου. Χρησιμοποιείστε το πρόγραμμα για να βρεθεί η ροπή αδρανείας της παρακάτω σύνθετης διατομής:



Λύση 1

Στον κώδικα της λύσης 2 του προηγούμενου παραδείγματος θα προσθέσουμε την κλάση της νέας διατομής την οποία ονομάζουμε `trig2()`.

```
# -*- coding: iso-8859-7 -*-
from math import sqrt
"Υπολογισμός κ.β. και ροπής αδρανείας σύνθετης διατομής."

class Diatomh:
    "Βασική διατομή για υπολογισμό ροπής αδρανείας."
    def __init__(self, b, h, xka, yka):
```

```
        "Καθορισμός των γεωμετρικών στοιχείων της διατομής."
        self.b = b
        self.h = h
        self.xka = xka
        self.yka = yka

    def ypolAktina(self):
        "Υπολογισμός ακτίνας αδρανείας διατομής."
        Ix = self.ypolAdran()
        A = self.ypolEmb()
        return sqrt(Ix/A)

class Orth(Diatomh):
    "Ορθογωνική διατομή για υπολογισμό ροπής αδρανείας."

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
        return self.xka + self.b/2, self.yka + self.h/2

    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.b *self.h**3 / 12

    def ypolEmb(self):
        "Υπολογισμός εμβαδού διατομής."
        return self.b * self.h

class Trig(Diatomh):
    "Τριγωνική διατομή για υπολογισμό ροπής αδρανείας."

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
        return self.xka + self.b/3, self.yka + self.h/3

    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.b *self.h**3 / 36

    def ypolEmb(self):
        "Υπολογισμός εμβαδού διατομής."
        return self.b * self.h / 2

class Trig2(Diatomh):
      "Τριγωνική διατομή με την ορθή γωνία κάτω δεξιά, για υπολογισμό ροπής
αδρανείας."

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
        return self.xka + 2*self.b/3, self.yka + self.h/3

    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.b *self.h**3 / 36

    def ypolEmb(self):
        "Υπολογισμός εμβαδού διατομής."
        return self.b * self.h / 2

def synkb(diats):
    "Υπολογισμός κ.β. σύνθετης διατομής."
    sx = sy = sA = 0
    for diat in diats:
        A = diat.ypolEmb()
        xc, yc = diat.ypolKb()
```

```
        sx += yc*A
        sy += xc*A
        sA += A
    return sy/sA, sx/sA

def synRopadr(diats, synxc, synyc):
    "Υπολογισμός ροπής αδρανείας σύνθετης διατομής."
    synIx = 0
    for diat in diats:
        A = diat.ypolEmb()
        xc, yc = diat.ypolKb()
        Ix = diat.ypolAdran()
        synIx += Ix + A * (yc-synyc)**2
    return synIx

def pyMain():
    "Standalone main program: example of inertia moments."
    diats = (  Trig2(0.35, 0.50, 0.00, 0.00),
               Orth(0.30, 0.50, 0.35, 0.00),
               Trig(0.35, 0.50, 0.65, 0.00),
            )
    xcs, ycs = synkb(diats)
    Ixs = synRopadr(diats, xcs, ycs)
    for diat in diats:
        xc, yc = diat.ypolKb()
        Ix = diat.ypolAdran()
        ix = diat.ypolAktina()
        print(xc, yc, Ix, ix)
    print("-------------------------------------------------------------------------
-------------")
    As = sum([diat.ypolEmb() for diat in diats])
    ixs = sqrt(Ixs/As)
    print(xcs, ycs, Ixs, ixs)

pyMain()
```

Παρατηρούμε ότι η εισαγωγή της νέας κλάσης `Trig2()` δεν επηρέασε καθόλου τις προηγούμενες κλάσεις, διότι όπως ειπώθηκε στο προηγούμενο παράδειγμα η κάθε κλάση είναι ανεξάρτητη. Επιπλέον παρατηρούμε ότι ο κώδικας που υπολογίζει τη σύνθετη διατομή δηλαδή οι συναρτήσεις `pyMain()`, `synkb()` και `synRopadr()` δεν άλλαξαν καθόλου (εκτός βέβαια από τα δεδομένα) και **παρέμεινε ο ίδιος ακριβώς κώδικας**.

Λύση 2 με παράγωγη κλάση και αντικατάσταση μεθόδου

Στον κώδικα της λύσης 1 παρατηρούμε ότι η κλάση trig2() έχει τις συναρτήσεις `ypolAdran()` και `ypolEmb()` ίδιες με αυτές την κλάσης `Trig()`. Συνεπώς θα ήταν χρήσιμο να κάνουμε την `Trig2()` παράγωγη της `Trig()` έτσι ώστε να αποφύγουμε επικάλυψη κώδικα, αλλά δημιουργούνται 2 ερωτήματα:
α) Η `Trig()` είναι ήδη παράγωγη της κλάσης `Diatomh()`. Στον αντικειμενοστραφή αυτό επιτρέπεται και έτσι η `Trig2()` κληρονομεί και όλες τις μεθόδους που ορίζονται στην κλάση `Trig()` και όλες τις μεθόδους που η `Trig()` κληρονόμησε από την κλάση `Diatomh()`.
β) Η κλάση `Trig2()` έχει διαφορετική της μέθοδο `ypolKb()` από την κλάση `Trig()`. Στον αντικειμενοστραφή αυτό αντιμετωπίζεται με τον ορισμό μεθόδου `ypolKb()` εκ νέου στην κλάση `Trig()`. Η μέθοδος αυτή αντικαθιστά (overrides) τη μέθοδο `ypolKb()` της κλάσης `Trig()`.
Παρακάτω δίνεται ο κώδικας για την `Trig2()`.

```
class Trig2(Trig):
      "Τριγωνική διατομή με την ορθή γωνία κάτω δεξιά, για υπολογισμό ροπής
αδρανείας."

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
        return self.xka + 2*self.b/3, self.yka + self.h/3
```

Όπως φαίνεται ο κώδικας της Trig2() έγινε πολύ μικρότερος. Ο υπόλοιπος κώδικας του προγράμματος δεν αλλάζει.

3. Υπολογισμός κέντρου βάρους και ροπής αδρανείας σύνθετης διατομής με αντικείμενα: Η σύνθετη διατομή ως αντικείμενο

Να συνταχθεί πρόγραμμα σε Python που να ορίζει τη σύνθετη διατομή ως αντικείμενο.

Λύση

Η σύνθετη διατομή που υπολογίζεται είναι και αυτή διατομή με κ.β., ροπή αδρανείας, εμβαδόν και ακτίνα αδρανείας. Συνεπώς και αυτή μπορεί να είναι μία κλάση που είναι παράγωγη της κλάσης Diatomh(). Ο ορισμός της σύνθετης διατομής δεν γίνεται με διαστάσεις, αλλά με ορισμό των στοιχειωδών διατομών που την αποτελούν. Αυτό σημαίνει ότι η μέθοδος __init__() πρέπει να είναι διαφορετική από τις άλλες κλάσεις, και έτσι η νέα κλάση της σύνθετης διατομής θα την αντικαθιστά.

```
# -*- coding: iso-8859-7 -*-
from math import sqrt
"Υπολογισμός κ.β. και ροπής αδρανείας σύνθετης διατομής."


class Diatomh:
    "Βασική διατομή για υπολογισμό ροπής αδρανείας."

    def __init__(self, b, h, xka, yka):
        "Καθορισμός των γεωμετρικών στοιχείων της διατομής."
        self.b = b
        self.h = h
        self.xka = xka
        self.yka = yka


    def ypolAktina(self):
        "Υπολογισμός ακτίνας αδρανείας διατομής."
        Ix = self.ypolAdran()
        A = self.ypolEmb()
        return sqrt(Ix/A)


class Orth(Diatomh):
    "Ορθογωνική διατομή για υπολογισμό ροπής αδρανείας."

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
        return self.xka + self.b/2, self.yka + self.h/2


    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.b *self.h**3 / 12


    def ypolEmb(self):
        "Υπολογισμός εμβαδού διατομής."
        return self.b * self.h


class Trig(Diatomh):
    "Τριγωνική διατομή για υπολογισμό ροπής αδρανείας."

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
```

```
        return self.xka + self.b/3, self.yka + self.h/3


    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.b *self.h**3 / 36


    def ypolEmb(self):
        "Υπολογισμός εμβαδού διατομής."
        return self.b * self.h / 2


class Trig2(Trig):
      "Τριγωνική διατομή με την ορθή γωνία κάτω δεξιά, για υπολογισμό ροπής
αδρανείας."

    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
        return self.xka + 2*self.b/3, self.yka + self.h/3


class Syntheth(Diatomh):
    "Σύνθετη διατομή αποτελούμενη από άλλες διατομές."

    def __init__(self, diats):
        "Καθορισμός των στοιχειωδών διατομών."
         self.diats = tuple(diats)    #Here we make a copy, in case diats is a
list instead of a tuple


    def ypolKb(self):
        "Υπολογισμός κ.β. σύνθετης διατομής."
        sx = sy = sA = 0
        for diat in self.diats:
            A = diat.ypolEmb()
            xc, yc = diat.ypolKb()
            sx += yc*A
            sy += xc*A
            sA += A
        return sy/sA, sx/sA


    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας σύνθετης διατομής."
        synxc, synyc = self.ypolKb()
        synIx = 0
        for diat in self.diats:
            A = diat.ypolEmb()
            xc, yc = diat.ypolKb()
            Ix = diat.ypolAdran()
            synIx += Ix + A * (yc-synyc)**2
        return synIx


    def ypolEmb(self):
        "Υπολογισμός εμβαδού σύνθετης διατομής."
        sA = 0
        for diat in self.diats:
            sA += diat.ypolEmb()
        return sA


def pyMain():
```

```
    "Standalone main program: example of inertia moments."
    diats = (  Trig2(0.35, 0.50, 0.00, 0.00),
               Orth(0.30, 0.50, 0.35, 0.00),
               Trig(0.35, 0.50, 0.65, 0.00),
            )
    syn = Syntheth(diats)
    diats = diats + (syn,)     #Νέα πλειάδα που έχει και τη σύνθετη διατομή στο
τέλος
    for diat in diats:
        xc, yc = diat.ypolKb()
        Ix = diat.ypolAdran()
        ix = diat.ypolAktina()
        print(xc, yc, Ix, ix)


pyMain()
```

Λύση με κρυφή αποθήκευση (caching) των στοιχείων της διατομής

Στην προηγούμενη λύση, παρατηρούμε ότι το εμβαδόν και το κ.β. όλων των στοιχειωδών διατομών υπολογίζονται ξανά από την αρχή σε κάθε μία από τις μεθόδους της κλάσης `syntheth()`. Επίσης αποθηκεύονται εντός της κλάσης όλες οι στοιχειώδεις διατομές. Για να αποφύγουμε αυτό το κόστος σε υπολογιστικό χρόνο και μνήμη, μπορούμε να υπολογίζουμε τα στοιχεία της διατομής (κ.β., ροπή αδρανείας κλπ) κατά την δημιουργία του στιγμιότυπου (instance) της κλάσης, και να τα αποθηκεύουμε σε μεταβλητές του στιμιότυπου. Έτσι όταν γίνει κλήση μία μεθόδου πχ της `ypolEmb()`, αυτή θα επιστρέφει το ήδη υπολογισμένο εμβαδόν.

```
class Syntheth(Diatomh):
    "Σύνθετη διατομή αποτελούμενη από άλλες διατομές."

    def __init__(self, diats):
        "Υπολογισμός των στοιχείων της σύβνθετης διατομής."
        sx = sy = sA = 0
        for diat in diats:
            A = diat.ypolEmb()
            xc, yc = diat.ypolKb()
            sx += yc*A
            sy += xc*A
            sA += A
        self.xc, self.yc, self.A = sy/sA, sx/sA, sA

        self.Ix = 0
        for diat in diats:
            A = diat.ypolEmb()
            xc, yc = diat.ypolKb()
            Ix = diat.ypolAdran()
            self.Ix += Ix + A * (yc-self.yc)**2


    def ypolKb(self):
        "Υπολογισμός κ.β. διατομής."
        return self.xc, self.yc


    def ypolAdran(self):
        "Υπολογισμός ροπής αδρανείας διατομής."
        return self.Ix


    def ypolEmb(self):
        "Υπολογισμός εμβαδού σύνθετης διατομής."
        return self.A
```

Οι υπόλοιπος κώδικας παραμένει ίδιος.