ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ
ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΠΟΛΙΤΙΚΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΔΟΜΟΣΤΑΤΙΚΗΣ

Ακαδ. Έτος: 2016-2017

Μάθημα: Εφαρμογές Η/Υ

Δευτέρα, 10/10/2016

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

Παραδείγματα για την 2η παράδοση – Εργαλεία ελέγχου ροής, σύνθετοι τύποι αντικειμένων

1. Υπολογισμός κέντρου βάρους και ροπής αδρανείας ορθογωνικής και τριγωνικής διατομής

Η κεντροβαρική ροπή αδρανείας Iy ορθογωνικής διατομής διαστάσεων b, h υπολογίζεται ως:

$$I_y = \frac{b h^3}{12}$$

και οι συντεταγμένες του κ.β.

$$x_c = x_{KA} + \frac{b}{2} \quad , \quad y_c = y_{KA} + \frac{h}{2}$$

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:

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

Να συνταχθεί πρόγραμμα σε Python που να υπολογίζει τη ροπή αδρανείας και το κ.β. με δεδομένα b, h, x_{KA}, y_{KA} και ανάλογα με τη μεταβλητή t που περιέχει “rect” ή “tria” για ορθογώνιο και τρίγωνο αντίστοιχα.

Λύση με “μονολιθικό κώδικα” (δηλαδή χωρίς συναρτήσεις ή αντικείμενα)

Σε ένα αρχείο Python (που τυπικά έχει την κατάληξη .py) αποτελεί και μία module (ενότητα) δηλαδή ένα σύνολο από εντολές ή συναρτήσεις ή αντικείμενα ή συνδυασμό αυτών. Η module έχει το όνομα του προθέματος του αρχείου, και για αυτό το αρχείο πρέπει να ονομάζεται με λατινικό γράμμα στην αρχή και στη συνέχεια λατινικά γράμματα, ψηφία ή την κάτω παύλα “_”. Όποιες εντολές υπάρχουν χωρίς να είναι σε μία συνάρτηση ή σε αντικείμενο εκτελούνται απευθείας, η μία μετά την άλλη από πάνω προς τα κάτω χωρίς να έχει προηγηθεί έλεγχος όλων των εντολών (εκτός από συντακτικά λάθη). Η πρώτη γραμμή μπορεί να περιέχει μία σειρά με κείμενο που εξηγεί συνοπτικά τι κάνει το πρόγραμμα που είναι στο αρχείο. Το κείμενο αυτό αποθηκεύεται στη μεταβλητή __doc__ της module.

```
"Computes the moments of inertia of orthogon or right triangle."
t = "trig"
b = 0.3
h = 0.5
xka = 0.8
yka = 0.6
if t == "orth":
    xc = xka + b/2
    yc = yka + h/2
    Ix = b*h**3/12
    A = b*h
else:
    xc = xka + b/3
    yc = yka + h/3
    Ix = b*h**3/36
    A = b*h/2
```

```
print(__doc__)
print(t, "A=", A, "I=", Ix, "xc=", xc, "yc=", yc)
```

Λύση με συναρτήσεις

Η χρήση συναρτήσεων βελτιώνει την αναγνωσιμότητα, αποτρέπει επανάληψη ίδιου κώδικα, προστατεύει από παρενέργειες όταν τα ονόματα δύο ανεξάρτητων μεταβλητών συμπίπτουν, και διευκολύνει τη διόρθωση αλλά και τη συντήρηση/επέκταση του προγράμματος. Αναπόφευκτα με χρήση συναρτήσεων ο κώδικας μεγαλώνει και για μικρά προγράμματα είναι κουραστική. Ένας εμπειρικός κανόνας είναι ότι αν ένα πρόγραμμα χωράει σε μία σελίδα οθόνης και ο κώδικας δεν θα χρησιμοποιηθεί από άλλο πρόγραμμα τότε μπορεί να γίνει χωρίς συναρτήσεις. Αλλιώς πρέπει να γίνει με συναρτήσεις. Η πρώτη γραμμή μίας συνάρτησης μπορεί να είναι κείμενο που εξηγεί συνοπτικά τι κάνει η συνάρτηση. Το κείμενο αυτό αποθηκεύεται στη μεταβλητή __doc__ της συνάρτησης. Ο ορισμός των συναρτήσεων πρπεί να γίνει πριν από τον κώδικα της module που τις χρησιμοποιεί.

```
"Computes the moments of inertia of a rectangle or right triangle, with
functions."

def orth(b, h, xka, yka):
    "Computes the moments of inertia of a rectangle."
    xc = xka + b/2
    yc = yka + h/2
    Ix = b*h**3/12
    A = b*h
    return (A, Ix, xc, yc)


def trig(b, h, xka, yka):
    "Computes the moments of inertia of a right triangle."
    xc = xka + b/3
    yc = yka + h/3
    Ix = b*h**3/36
    A = b*h/2
    return A, Ix, xc, yc


t = "orth"
b = 0.3
h = 0.5
xka = 0.8
yka = 0.6
if t == "orth":
    print(orth.__doc__)
    A, Ix, xc, yc = orth(b, h, xka, yka)
else:
    print(trig.__doc__)
    (A, Ix, xc, yc) = trig(b, h, xka, yka)
print(__doc__)
print(t, "A=", A, "I=", Ix, "xc=", xc, "yc=", yc)
```

Οι παρενθέσεις γύρω από τα ορίσματα που επιστρέφονται από τη συνάρτηση δεν είναι υποχρεωτικές (στην εντολή return και όταν καλείται η συνάρτηση). Φαίνεται ότι μία συνάρτηση μπορεί να επιστρέψει πολλά ορίσματα αλλά στην πραγματικότητα η συνάρτηση επιστρέφει μία πλειάδα η οποία αποτελείται από τα ορίσματα που επιστρέφονται.

Λύση με συναρτήσεις

Στην προηγούμενη λύση υπάρχει κώδικας που δεν είναι ενσωματωμένος (encapsulated) σε κάποια συνάρτηση ή αντικείμενο, ο οποίος λέγεται κώδικας επιπέδου module. Αυτός ο κώδικας μπορεί να γίνει πηγή λαθών, καθώς οι μεταβλητές που ορίζονται στο επίπεδο της module είναι σφαιρικές (global) και τις βλέπουν όλες οι συναρτήσεις και τα αντικείμενα. Είναι καλή προγραμματιστική αυτός ο κώδικας να ενσωματώνεται σε μία “κύρια” συνάρτηση. Παρεμπιπτόντως, η κύρια συνάρτηση έχει συνήθως επιτελικό χαρακτήρα, δηλαδή δεν κάνει κάποια εργασία από μόνη της, αλλά καλεί άλλες συναρτήσεις που κάνουν τις

απαραίτητες εργασίες. Ο κώδικας σε επίπεδο module περιορίζεται στην κλήση της κύριας συνάρτησης.

```
"Computes the moments of inertia of a rectangle or right triangle, with
functions, main function."

def orth(b, h, xka, yka):
    "Computes the moments of inertia of a rectangle."
    xc = xka + b/2
    yc = yka + h/2
    Ix = b*h**3/12
    A = b*h
    return (A, Ix, xc, yc)


def trig(b, h, xka, yka):
    "Computes the moments of inertia of a right triangle."
    xc = xka + b/3
    yc = yka + h/3
    Ix = b*h**3/36
    A = b*h/2
    return A, Ix, xc, yc


def pyMain():
    "Standalone main program: example of inertia moments."
    t = "orth"
    b = 0.3
    h = 0.5
    xka = 0.8
    yka = 0.6
    if t == "orth":
        A, Ix, xc, yc = orth(b, h, xka, yka)
    else:
        (A, Ix, xc, yc) = trig(b, h, xka, yka)
    print(t, "A=", A, "I=", Ix, "xc=", xc, "yc=", yc)


pyMain()
```

Αν παραληφθεί η κλήση της pyMain() στον κώδικα επιπέδου module, μία εκτέλεση του προγράμματος δεν θα κάνει απολύτως τίποτα (εκτός από την επισήμανση συντακτικών λαθών αν υπάρχουν). Αυτό το χαρακτηριστικό δίνει τη δυνατότητα να γίνει η module βιβλιοθήκη (ή πακέτο), δηλαδή να παρέχει κάποιες συναρτήσεις που μπορεί να καλέσουν άλλες modules μέσω της εντολής import.

2. Υπολογισμός φερτών υλών σε λιμνοδεξαμενή

Ο ετήσιος όγκος φερτών υλών V_φ (m^3) σε λεκάνη απορροής εμβαδού A (km^2) προσεγγίζεται ως:

$$V_\varphi = Q_s A \quad , \quad Q_s = 1098\, A^{-0.24}$$

Από τον ετήσιο όγκο V_φ στη λιμνοδεξαμενή παρακρατείται (αποτίθεται) ποσοστό p (%) των φερτών. Θεωρώντας V_L (m^3) τον όγκο νερού στη λιμνοδεξαμενή και Q_{in} (m^3) τον ετήσιο όγκο νερού που εισρέει στη δεξαμενή, ο συντελεστής K απόθεσης υπολογίζεται:

$$K = \frac{V_L}{Q_{in}}$$

Το ποσοστό απόθεσης υπολογίζεται:

$p = 100$ αν K>30

$p = 100 - (26\, K^{-0.2} - 12)$ αν K≤30

Να συνταχθεί πρόγραμμα σε Python που να διαβάζει τα A, Q_{in}, V_L και N και να υπολογίζει τον συνολικό όγκο απόθεσης φερτών μετά από N έτη, θεωρώντας:

α) Σταθερό ετήσιο όγκο απόθεσης σε όλα τα έτη

β) Μεταβλητό ετήσιο όγκο απόθεσης που υπολογίζεται από τον όγκο νερού στη λίμνη που απομένει στη

λίμνη, αν αφαιρεθεί από τον όγκο του νερού της λιμνοδεξαμενης V_L ο όγκος των φερτών που αποτίθενται. Δοκιμάστε το πρόγραμμα με A=32.9 km^2, $Q_{in}=9\cdot10^6$ m^3, $V_L=6.5\cdot10^6$ m^3 και N=100 έτη.

Λύση

Θα δημιουργηθούν συναρτήσεις που υπολογίζουν το ετήσιο όγκο απόθεσης, το συνολικό όγκο απόθεσης με σταθερό ετήσιο, το συνολικό όγκο απόθεσης με μεταβλητό ετήσιο και τέλος μία κύρια συνάρτηση που διαβάζει τα δεδομένα και υπολογίζει τα αποτελέσματα.

```
# -*- coding: iso-8859-7 -*-
"Υπολογισμός φερτών υλών."

def ethsFertes(A, Qin, VL):
    "Ετήσιος όγκος απόθεσης φερτών υλών."
    Qs = 1098 * A**(-0.24)
    Vf = Qs * A
    K = VL/Qin
    if K > 30:
        p = 100
    else:
        p = 100 - (26 * K**(-0.2) - 12)
    Vfk = Vf * p / 100
    return Vfk


def fertes1(A, Qin, VL, N):
    "Όγκος φερτών σε N έτη με σταθερό ετήσιο όγκο απόθεσης."
    return N*ethsFertes(A, Qin, VL)


def fertes2(A, Qin, VL, N):
    "Όγκος φερτών σε N έτη με σταθερό ετήσιο όγκο απόθεσης."
    sVfk = 0
    for i in range(N):
        Vfk = ethsFertes(A, Qin, VL-sVfk)
        sVfk += Vfk
    return sVfk


def pyMain():
    "Διάβαζει δεδομένα και κάνει υπολογισμούς."
    A   = float(input("Εμβαδόν λεκάνης απορροής   (km2):"))
    Qin = float(input("Ετήσιος όγκος εισροής νερού (m3):"))
    VL  = float(input("Όγκος νερού λιμνοδεξαμενής  (m3):"))
    N   =   int(input("Έτη λειτουργία λιμνοδεξαμενής   :"))
    print("Ετήσιος όγκος απόθεσης (m3):", ethsFertes(A, Qin, VL))
    print("Συνολικός όγκος απόθεσης με σταθερό   ετήσιο (m3):", fertes1(A, Qin,
VL, N))
    print("Συνολικός όγκος απόθεσης με μεταβλητό ετήσιο (m3):", fertes2(A, Qin,
VL, N))


pyMain()
```

Αν το αρχείο περιέχει μη λατινικούς χαρακτήρες, όπως για παράδειγμα ελληνικά, η Python απαιτεί να οριστεί στην πρώτη ή τη δεύτερη σειρά του αρχείου ποια κωδικοσελίδα (codepage) χρησιμοποιείται, πχ `-*- coding: iso-8859-7 -*-`. Για παράδειγμα τα ελληνικά μπορεί να είναι ελληνικά για το παλιό DOS (CP737), για παλιά Windows (ISO-8859-7) ή κάποια μορφής unicode που ενοποιεί όλες τις γλώσσες του κόσμου, όπως η UTF-8. Αν η κωδικοσελίδα είναι UTF-8, δεν χρειάζεται να οριστεί.

3. Υπολογισμός κέντρου βάρους και ροπής αδρανείας σύνθετης διατομής

Να συνταχθεί πρόγραμμα σε Python που να υπολογίζει τη ροπή αδρανείας σύνθετης διατομής που αποτελείται από στοιχειώδεις ορθογωνικές και τριγωνικές διατομές με τα δεδομένα του παραδείγματος 1. Το

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

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

Λύση με πλειάδες σε κάθε μεταβλητή

Αφού υπάρχουν πολλές διατομές σημαίνει ότι θα υπάρχουν πολλά b, πολλά h, πολλά t κλπ. Μία λύση είναι να ορίσουμε κάθε μεταβλητή t, b, h, xka, yka ως πλειάδα (tuple), που κάθε μία να έχεσι τόσα στοιχεία όσες και διατομές. Αυτό δημιουργεί το ερώτημα αν όλες οι πλειάδες έχουν τον ίδιο πλήθος στοιχείων. Η εντολή assert επιτρέπει να ελέγξουμε αν αυτό συμβαίνει, και αν δεν συμβαίνει η εντολή assert σταματάει το πρόγραμμα με μήνυμα λάθους. Η εντολή assert βοηθάει στην εκσφαλμάτωση του προγράμματος (debugging), αλλά δεν πρέπει να χρησιμοποιείται σε εμπορικά προγράμματα. Οι συναρτήσεις orth() και trig() έχουν ληφθεί από το προηγούμενο παράδειγμα.

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

def orth(b, h, xka, yka):
    "Computes the moments of inertia of a rectangle."
    xc = xka + b/2
    yc = yka + h/2
    Ix = b*h**3/12
    A = b*h
    return (A, Ix, xc, yc)


def trig(b, h, xka, yka):
    "Computes the moments of inertia of a right triangle."
    xc = xka + b/3
    yc = yka + h/3
    Ix = b*h**3/36
    A = b*h/2
    return A, Ix, xc, yc


def synkb(t, b, h, xka, yka):
    "Υπολογισμός κ.β. σύνθετης διατομής."
    sx = sy = sA = 0
    for i in range(len(t)):
        if t[i] == "orth":
            A, Ix, xc, yc = orth(b[i], h[i], xka[i], yka[i])
        else:
            (A, Ix, xc, yc) = trig(b[i], h[i], xka[i], yka[i])
        sx += yc*A
        sy += xc*A
        sA += A
    return sy/sA, sx/sA


def synRopadr(t, b, h, xka, yka, synxc, synyc):
    "Υπολογισμός ροπής αδρανείας σύνθετης διατομής."
    synIx = 0
    for i in range(len(t)):
        if t[i] == "orth":
            A, Ix, xc, yc = orth(b[i], h[i], xka[i], yka[i])
        else:
            (A, Ix, xc, yc) = trig(b[i], h[i], xka[i], yka[i])
        synIx += Ix + A * (yc-synyc)**2
    return synIx
```

```
def pyMain():
    "Standalone main program: example of inertia moments."
    t = ("orth", "trig")
    b = (0.3, 0.5)
    h = 0.5, 0.5
    xka = (0, 0.3)
    yka = 0, 0
    n = len(t)
    assert len(b) == n
    assert len(h) == n
    assert len(xka) == n
    assert len(yka) == n
    xc, yc = synkb(t, b, h, xka, yka)
    Ix = synRopadr(t, b, h, xka, yka, xc, yc)
    print("κ.β. σύνθετης διατομής:", xc, yc)
    print("Ix   σύνθετης διατομής:", Ix)


pyMain()
```

Λύση με πλειάδες από πλειάδες

Η Python επιτρέπει μία πλειάδα να έχει ετερογενή στοιχεία (πχ πραγματικούς και κείμενο) και αυτή η δυνατότητα επιτρέπει να ορίσουμε τα στοιχεία μίας διατομής όλα σε μία πλειάδα, και στη συνέχεια να έχουμε μία πλειάδα από τέτοιες πλειάδες για όλες τις διατομές. Έτσι αποφεύγεται ο κίνδυνος η κάθε πλειάδα να έχει διαφορετικό πλήθος στοιχείων και ο κώδικας γίνεται πολύ πιο ευανάγνωστος. Οι συναρτήσεις orth() και trig() παραμένουν ίδιες και παραλείπονται.

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

def synkb(diats):
    "Υπολογισμός κ.β. σύνθετης διατομής."
    sx = sy = sA = 0
    for diat in diats:
        t, b, h, xka, yka = diat
        if t == "orth":
            A, Ix, xc, yc = orth(b, h, xka, yka)
        else:
            A, Ix, xc, yc = trig(b, h, xka, yka)
        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 t, b, h, xka, yka in diats:
        if t == "orth":
            A, Ix, xc, yc = orth(b, h, xka, yka)
        else:
            A, Ix, xc, yc = trig(b, h, xka, yka)
        synIx += Ix + A * (yc-synyc)**2
    return synIx


def pyMain():
    "Standalone main program: example of inertia moments."
    diats = (  ("orth", 0.3, 0.5, 0,   0), \
               ("trig", 0.5, 0.5, 0.3, 0), \
            )
```

```
    xc, yc = synkb(diats)
    Ix = synRopadr(diats, xc, yc)
    print("κ.β. σύνθετης διατομής:", xc, yc)
    print("Ix   σύνθετης διατομής:", Ix)


pyMain()
```

Για να συνεχιστεί μία εντολή στην επόμενη σειρά πρέπει να γραφεί η ανάποδη κάθετος στο τέλος της σειράς. Η ανάποδη κάθετος δεν είναι απαραίτητη αν σε μία σειρά δεν έχει τοποθετηθεί η δεξιά παρένθεση ή αγκύλη ή άγκιστρο (δηλαδή οι παρενθέσεις ή αγκύλες ή άγκιστρα κλείνουν στην επόμενη σειρά).

Λύση με πλειάδες και λίστες από πλειάδες

Παρατηρούμε ότι η συνάρτηση synRopadr() υπολογίζει πάλι το κ.β. και τη ροπή αδρανείας κάθε διατομής, παρόλο που οι υπολογισμοί έχουν ξαναγίνει στη συνάρτηση synkb(). Για οικονομία χρόνου υπολογισμούς (αλλά εις βάρος της μνήμης) μπορούμε να κρατούμε τις υπολογισμένες τιμές κάθε διατομής σε μία πλειάδα, και τις τιμές όλων των διατομών σε μία λίστα από πλειάδες. Όπως φαίνεται στον κώδικα μπορούμε να πάρουμε όλες τις μεταβλητές εξόδου μίας συνάρτησης ως μία πλειάδα. Επίσης, η συνάρτηση synRopadr() απλοποιείται πολύ με αυτόν τον τρόπο. Πάλι οι συναρτήσεις orth() και trig() παραμένουν ίδιες και παραλείπονται.

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

def synkb(diats):
    "Υπολογισμός κ.β. σύνθετης διατομής."
    sx = sy = sA = 0
    aic = []
    for t, b, h, xka, yka in diats:
        if t == "orth":
            times = orth(b, h, xka, yka)
        else:
            times = trig(b, h, xka, yka)
        A, Ix, xc, yc = times
        sx += yc*A
        sy += xc*A
        sA += A
        aic.append(times)
    return sy/sA, sx/sA, aic


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


def pyMain():
    "Standalone main program: example of inertia moments."
    diats = (  ("orth", 0.3, 0.5, 0,   0), \
               ("trig", 0.5, 0.5, 0.3, 0), \
            )
    xc, yc, aic = synkb(diats)
    Ix = synRopadr(aic, xc, yc)
    print("κ.β. σύνθετης διατομής:", xc, yc)
    print("Ix   σύνθετης διατομής:", Ix)


pyMain()
```

3. Υπολογισμός βλετιστης διατομής οπλισμού

Το εμβαδόν διατομής σε cm^2 ράβδου οπλισμού διαμέτρου Φ mm δίνεται από:

$$A_\Phi = \frac{\pi \Phi^2}{400}$$

Οπλισμός δοκού απαιτούμενου εμβαδού A_s κατανέμεται σε n_Φ ράβδους ως:

$$n_\Phi = \left\lceil \frac{A_s}{A_\Phi} \right\rceil \quad , \quad n_\Phi \geq 2$$

όπου οι αγκύλες σημαίνουν στρογγύλευση προς τα πάνω. Η σχέση αυτή υποδηλώνει ότι ο πραγματικός οπλισμός $A_{s,act}$ είναι μεγαλύτερος ή το πολύ ίσος από τον απαιτούμενο A_s εις βάρος της οικονομίας:

$$A_{s,act} = n_\Phi A_\Phi \geq A_s$$

Να συνταχθεί πρόγραμμα σε Python που να υπολογίζει ποια διάμετρος από τις Φ12, Φ14, Φ16, Φ18, Φ20, Φ22, Φ25, Φ30 δίνει τον ελάχιστο οπλισμό $A_{s,act}$ ($\geq A_s$).

Λύση με λίστες από πλειάδες

Για κάθε διάμετρο Φ φτιάχνουμε μία πλειάδα ($A_{s,act}$, Φ, n_Φ) και αποθηκεύουμε όλες τις πλειάδες σε μία λίστα. Στην συνέχεια βρίσκουμε την πλειάδα με το μικρότερο $A_{s,act}$ και το δεύτερο στοιχείο τηξς πλειάδας είναι η ζητούμενη διάμετρος.

```
# -*- coding: iso-8859-7 -*-
"Υπολογισμός βέλτιστης διαμέτρου οπλισμού."
from math import pi, ceil

def bars(As, phi):
    "Υπολογλιζει το πλήθος ράβδων που απαιτούνται για συνολικό οπλισμό As."
    Aphi = pi*phi**2/400
    nphi = ceil(As/Aphi)
    if nphi < 2: nphi = 2
    Asact = nphi*Aphi
    return Asact, phi, nphi


As = 12
afn = []
for phi in (12, 14, 16, 18, 20, 22, 25, 30):
    afn1 = bars(As, phi)
    afn.append(afn1)
afn1 = min(afn)
Asact, phi, nphi = afn1
print("Βέλτιστοι ράβδοι", nphi, "Φ", phi, "    As,act=", Asact)"Υπολογισμός κ.β.
και ροπής αδρανείας σύνθετης διατομής."
```

Λύση με έμμεσο βρόχο

Η λίστα μπορεί να δημιουργηθεί με έμμεσο βρόχο (list comprehension). Η συνάρτηση bars() παραμένει ίδια και παραλείπεται.

```
# -*- coding: iso-8859-7 -*-
"Υπολογισμός βέλτιστης διαμέτρου οπλισμού."
from math import pi, ceil

As = 12
afn1 = min([bars(As, phi) for phi in (12, 14, 16, 18, 20, 22, 25, 30)])
Asact, phi, nphi = afn1
print("Βέλτιστοι ράβδοι", nphi, "Φ", phi, "    As,act=", Asact)
```

Λύση με έμμεσο βρόχο 2

Ο κώδικας μπορεί να συμπυκνωθεί λίγο περισσότερο και να γίνει ακόμα πιο ευανάγνωστος. Η συνάρτηση bars() παραμένει ίδια και παραλείπεται.

```
# -*- coding: iso-8859-7 -*-
"Υπολογισμός βέλτιστης διαμέτρου οπλισμού."
from math import pi, ceil

As = 12
Asact, phi, nphi = min([bars(As, phi) for phi in (12, 14, 16, 18, 20, 22, 25,
30)])
print("Βέλτιστοι ράβδοι", nphi, "Φ", phi, "    As,act=", Asact)
```

Ο παραπάνω κώδικας δείχνει τις δυνατότητες της Python, να μπορεί με λίγες γραμμές ευανάγνωστου κώδικα να κάνει μπορεί να συμπυκνωθεί λίγο περισσότερο και να γίνει ακόμα πιο ευανάγνωστος. Ο παραπάνω κώδικας μπορεί να απλουστευθεί ακόμη περισσότερο με κατάργηση της λίστας όπως θα πούμε σε επόμενο μάθημα:

```
Asact, phi, nphi = min(bars(As, phi) for phi in (12, 14, 16, 18, 20, 22, 25,
30))
```