Σημειωματάριο Δευτέρας 30 Οκτ. 2017

Συναρτήσεις (functions)

Μια συνάρτηση στην Python είναι κομμάτι κώδικα που φέρει το δικό του όνομα (ακολουθεί τη λέξη κλειδί `def` στον ορισμό της συνάρτησης, έχει τα δικά της ορίσματα ή παραμέτρους (arguments) και όταν τελειώνει η εκτέλεσή του επιστρέφει στο κομμάτι του κώδικα που την "κάλεσε" (με την εντολή `return`).

Όπως βλέπουμε στα παραδείγματα παρακάτω ο ορισμός μιας συνάρτησης αρχίζει με μια γραμμή της μορφής

```
def ΌνομαΣυνάρτησης ( ΌνομαΟρίσματος1, ΌνομαΟρίσματος2, ...):
```

Ακολουθεί το κείμενο (ορισμός) της συνάρτησης γραμμένο πιο μέσα (indented) με τρόπο ώστε να υπάγεται στο `def`.

Στα παρακάτω παραδείγματα θα δούμε πώς φτιάχνουμε μια συνάρτηση και πώς τη χρησιμοποιούμε. Ο κύριος λόγος για τον οποίο χρησιμοποιούμε συναρτήσεις είναι για να μην επαναλαμβάνουμε ξανά και ξανά τα ίδια κομμάτια κώδικα. Μάλιστα αν, γράφοντας ένα πρόγραμμα, αντιληφθούμε ότι γράφουμε και δεύτερη φορά το ίδιο κομμάτι κώδικα τότε θα πρέπει να αναρωτηθούμε αν είναι μήπως καλύτερα αυτό το κομμάτι να το πακετάρουμε σε μια συνάρτηση την οποία μετά εύκολα θα καλούμε εκεί όπου τη χρειαζόμαστε.

Εύρεση ρίζας μια συνάρτησης με τη μέθοδο της διχοτόμησης

Αν έχουμε μια συνάρτηση $f : [a, b] \to \mathbb{R}$ που είναι συνεχής και έχει ετερόσημες τιμές στα άκρα του διαστήματος τότε, από το θεώρημα ενδιάμεσης τιμής, η συνάρτηση f έχει σίγουρα ρίζα (σημείο μηδενισμού) μέσα στο διάστημα $[a, b]$.

Ένας τρόπος να υπολογίσουμε προσεγγιστικά μια τέτοια ρίζα είναι η λεγόμενη μέθοδος της διχοτόμησης. Σε κάθε βήμα της μεθόδου έχουμε ένα διάστημα στα άκρα του οποίου η συνάρτηση παίρνει ετερόσημες τιμές. Υπολογίζουμε την τιμή στο μέσο του διαστήματος και ελέγχουμε τα δύο μισά διαστήματα για το αν έχουν επίσης την ιδιότητα των ετερόσημων άκρων. Αν δεν την έχει το αριστερό μισό διάστημα τότε σίγουρα την έχει το δεξί. Επιλέγουμε ένα από τα δύο μισά διαστήματα που έχει την ιδιότητα και συνεχίζουμε τον υποδιπλασιασμό του διαστήματος έως ότου το διάστημα που έχουμε να είναι τόσο μικρό όσο και η προσέγγιση της ρίζας που είμαστε διατεθειμένοι να δεχτούμε. Τότε σταματάμε και αναφέρουμε το ένα από τα δύο άκρα (π.χ. το αριστερό) ως ρίζα του διαστήματος.

Στο επόμενο υλοποιούμε τη μέθοδο αυτή για τη συνάρτηση

$$f(x) = (x - 1)^3 - \frac{1}{2},$$

στο διάστημα $[1, 2]$, η οποία είναι συνεχής και έχει ετερόσημες τιμές στα δύο άκρα.

In [1]:
```
a = 1; b = 2 # Τα άκρα του διαστήματος όπου ψάχνουμε τη ρίζα. Τα a και b θα παριστάνουν κάθε φορά
             # το διάστημα στο οποίο έχουμε εγκλωβίσει τη ρίζα.

while b-a > 1e-6: # Όσο το διάστημα είναι >= 1e-6 (αυτή είναι η ακρίβεια προσέγγισης της ρίζας που επιθυμ
ούμε)
    m = (a+b)/2 # m είναι το μέσο του διαστήματος
    fa = (a-1)**3-0.5 # Η τιμή της συνάρτησης στο a
    fm = (m-1)**3-0.5 # Η τιμή της συνάρτησης στο b
    if fa*fm <= 0: # Αν είναι ετερόσημες
        b = m # μεταφέρουμε το b στο m και συνεχίζουμε με το αριστερό μισό διάστημα
    else: # αλλιώς
        a = m # μεταφέρουμε το a στο m και συνεχίζουμε με το δεξί μισό διάστημα
print("Η ρίζα είναι το {}".format(a))
```

```
Η ρίζα είναι το 1.7937002182006836
```

Παρατηρείστε ότι αν στο παραπάνω πρόγραμμα θέλουμε να αλλάξουμε τη συνάρτηση τότε πρέπει να τροποποιήσουμε δύο γραμμές του κώδικα με πολύ όμοιο τρόπο (τους ορισμούς των `fa` και `fm`).

Αντί γι' αυτό γράφουμε τη συνάρτησή μας ως μια συνάρτηση python με όνομα και πάλι `f` (θα μπορούσε όμως να είναι οτιδήποτε) και μέσα στον αλγόριθμο διχοτόμησης χρησιμοποιούμε κλήσεις προς τη συνάρτηση αυτή αντί να γράφουμε τον τύπο της συνάρτησης μέσα στον αλγόριθμο διχοτόμησης. Έτσι και ο αλγόριθμος διχοτόμησης είναι γραμμένος με πιο καθαρό τρόπο και φαίνεται που ακριβώς υπολογίζεται η συνάρτησή μας, όποια κι αν είναι αυτή, αλλά και η αλλαγή της συνάρτησης γίνεται πιο εύκολα, σε ένα και μόνο σημείο, μέσα στον ορισμό της συνάρτησης python `f` (ήδη το κάναμε και αλλάξαμε τη συνάρτηση στην $f(x) = (x-1)^5 - \frac{1}{2}$.

In [2]:
```
a = 1; b = 2

def f(x): # Επικεφαλίδα ορισμού της συνάρτησης f. Το x είναι η μοναδική παράμετρος (όρισμα) της συνάρτηση
ς
    return (x-1)**5-0.5 # Δοθέντος του x επιστρέφουμε την τιμή που θέλουμε να έχει η συνάρτησή μας στο x.

while b-a > 1e-10:
    m = (a+b)/2
    fa = f(a) # υπολογίζουμε τη συνάρτηση στο αριστερό άκρο
    fm = f(m) # και στο μέσο
    if fa*fm <= 0:
        b = m
    else:
        a = m
print("Η ρίζα είναι το {}".format(a))
```

```
Η ρίζα είναι το 1.8705505632678978
```

Στο επόμενο πακετάρουμε και τον αλγόριθμο διχοτόμησης σε μια συνάρτηση `findroot`, με παραμέτρους την `f` (που οφείλει να είναι μια συνάρτηση python η οποία να επιστρέφει για κάθε πραγματικό αριθμό x τον αριθμό $f(x)$), το `a` και το `b` (που είναι το αριστερό και δεξί άκρο του διαστήματος όπου ψάχνουμε τη ρίζα). Η συνάρτηση `findroot` μας επιστρέφει τη ρίζα που βρήκε.

Έπειτα καλούμε τη συνάρτηση `findroot` με δύο διαφορετικές μαθηματικές συναρτήσεις που έχουμε ορίσει παραπάνω, τις `f` και `f1`, δίνοντας απλά το όνομα της συνάρτησης της οποίας ρίζα ψάχνουμε. Ο κόπος που γλυτώνουμε είναι φανερός. Δε χρειάζεται να γράψουμε δύο φορές τον αλγόριθμο διχοτόμησης, μια φορά για κάθε συνάρτηση. Τον γράφουμε μια φορά ως τη συνάρτηση `findroot` την οποία καλούμε όσες φορές θέλουμε, με όποιες συναρτήσεις θέλουμε και όποια άκρα διαστημάτων θέλουμε.

In [6]:
```
def f(x):
    return (x-1)**3-0.5

def f1(x):
    return (x-1)**5-0.5

def findroot(f, a, b): # Επαναλαμβάνουμε εδώ ουσιαστικά τη μέθοδο διχοτόμησης όπως παραπάνω αλλά επιστρέφουμε
    while b-a > 1e-10: # τι ρίζα που βρήκαμε με το return παρακάτω.
        m = (a+b)/2
        fa = f(a)
        fm = f(m)
        if fa*fm <= 0:
            b = m
        else:
            a = m
    return a

r = findroot(f, 1, 2) # Εδώ βρίσκουμε ρίζα της συνάρτησης f
print("Η ρίζα είναι το {}".format(r))

r = findroot(f1, 1, 2) # Εδώ βρίσκουμε ρίζα της συνάρτησης f1
print("Η ρίζα είναι το {}".format(r))
```

```
Η ρίζα είναι το 1.7937005259445868
Η ρίζα είναι το 1.8705505632678978
```

Το επόμενο πρόβλημα που κοιτάμε είναι το εξής. Μας δίνεται ένα σύνολο από σημεία στο επίπεδο

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

και ένα ακόμη σημείο (x, y) και θέλουμε να βρούμε την απόσταση του σημείου (x, y) από το σύνολο σημείων. Με άλλα λόγια θέλουμε να βρούμε την ελάχιστη τιμή της ποσότητας

$$dist((x, y), (x_j, y_j)), \quad \text{όταν } j = 1, 2, \ldots, N.$$

Εδώ $dist\big((x, y), (z, w)\big)$ είναι η απόσταση των σημείων (x, y) και (x, w) η οποία γνωρίζουμε ότι δίνεται από τον τύπο (Πυθαγόρειο θεώρημα):

$$dist((x, y), (z, w)) = ((x - z)^2 + (y - w)^2)^{1/2}.$$

Τα σημεία μας τα έχουμε σε μια λίστα L ως λίστες μήκους 2 το καθένα, και το μοναδικό μας σημείο ως τις μεταβλητές x και y.

In [7]:
```
import math # Το χρειαζόμαστε για τον υπολογισμό της τετραγωνικής ρίζας

L = [ [1, 2], [0.1, -2], [3, 4], [-10, -3] ] # Η λίστα με το σύνολο των σημείων

x, y = 0.1, 3 # Το σημείο του οποίου ψάχνουμε την απόσταση από τα σημεία της L

D = [] # Εδώ αποθηκεύουμε τις αποστάσεις
for p in L: # παίρνουμε τις αποστάσεις του (x,y) από κάθε σημείο p της λίστας
    d = math.sqrt((x-p[0])**2 + (y-p[1])**2) # αυτή είναι η απόσταση του (x, y) από το p
    D.append(d) # τη βάζουμε στη λίστα των αποστάσεων

mindistance = min(D) # υπολογίζουμε το ελάχιστο στοιχείο της λίστας D

print("Η ελάχιστη απόσταση είναι {}".format(mindistance))
```

```
Η ελάχιστη απόσταση είναι 1.3453624047073711
```

Τώρα πακετάρουμε τον προηγούμενο αλγόριθμο σε μια συνάρτηση `mindistance(q, L)` όπου q είναι το σημείο την απόσταση του οποίου ψάχνουμε και L είναι η λίστα των σημείων. Η συνάρτηση επιστρέφει την ελάχιστη από τις αποστάσεις του q από τα σημεία της L.

In [8]:
```
import math

def mindistance(q, L):
    D = []
    for p in L:
        d = math.sqrt((q[0]-p[0])**2 + (q[1]-p[1])**2)
        D.append(d)
    return min(D)

md = mindistance([0.1, 3], [ [1, 2], [0.1, -2], [3, 4], [-10, -3] ]) # Εδώ καλούμε τη συνάρτηση με
                                    # q = [0.1, 3] και L = [ [1, 2], [0.1, -2], [3, 4], [-10, -3] ]

print("Η ελάχιστη απόσταση είναι {}".format(md))
```

```
Η ελάχιστη απόσταση είναι 1.3453624047073711
```

Στο επόμενο πρόγραμμα χρησιμοποιούμε τη συνάρτηση `mindistance(q, L)` για να υπολογίσουμε, με τη συνάρτηση `minpair(L)`, την ελάχιστη απόσταση ανάμεσα σε δύο σημεία της λίστας L. Η συνάρτηση `minpair(L)` επιστρέφει την ελάχιστη απόσταση ανάμεσα σε δύο σημεία της L. Το κάνει αυτό ως εξής: για κάθε σημείο της L βρίσκει την απόστασή του από τα υπόλοιπα σημεία της L χρησιμοποιώντας τη συνάρτηση `mindistance`. Τέλος βρίσκει τον ελάχιστο από αυτούς τους αριθμούς.

In [9]:
```
import math

def mindistance(q, L): # Είναι ίδια όπως και στο προηγούμενο πρόγραμμα
    D = []
    for p in L:
        d = math.sqrt((q[0]-p[0])**2 + (q[1]-p[1])**2)
        D.append(d)
    return min(D)

def minpair(L): # Επιστρέφει την ελάχιστη απόσταση ανάμεσα σε δύο σημεία της λίστας L (έχει την ίδια μορφή όπως πριν)
    D = [] # σε αυτή τη λίστα κρατάμε όλες τις αποστάσεις ενός σημείου από όλα τα υπόλοιπα
    for i in range(len(L)): # για κάθε σημείο της λίστας L
        d = mindistance(L[i], L[:i]+L[i+1:]) # βρίσκουμε την απόστασή του από τα υπόλοιπα σημεία
            # Η λίστα L[:i]+L[i+1:] περιέχει όλα τα σημεία της L εκτός από το L[i]
        D.append(d)
    return min(D) # επιστρέφουμε την ελάχιστη τέτοια απόσταση (σημείου προς όλα τα υπόλοιπα)

X = [ [0, 0], [1, 0], [1, 1], [0, 1], [0.5, 0.6] ] # αυτά είναι τα σημεία από τα οποία ψάχνουμε το πλησιέστερο ζεύγος

mp = minpair(X) # καλούμε τη συνάρτηση που μόλις γράψαμε

print("Η ελάχιστη απόσταση ζεύγους είναι {}".format(mp))
```

```
Η ελάχιστη απόσταση ζεύγους είναι 0.6403124237432849
```

Στο επόμενο πρόγραμμα τροποποιούμε τη συνάρτηση `mindistance` στη συνάρτηση `newmindistance` ώστε όχι μόνο να μας βρίσκει την ελάχιστη απόσταση του σημείου q από όλα τα σημεία της L αλλά να μας λέει και από ποιο σημείο της L είναι αυτή η απόσταση (η ελάχιστη απόσταση μπορεί βέβαια να "πιάνεται" σε παραπάνω από ένα σημείο της L αλλά η συνάρτησή μας επιστρέφει ένα από αυτά).

Η συνάρτησή μας επιστρέφει τώρα μια λίστα δύο στοχείων. Το πρώτο στοιχείο της λίστας είναι η ελάχιστη απόσταση και το δεύτερο είναι η θέση στην L ενός σημείου όπου υλοποιείται η ελάχιστη απόσταση.

In [14]:
```
import math

def newmindistance(q, L):
    D = []
    for p in L:
        d = math.sqrt((q[0]-p[0])**2 + (q[1]-p[1])**2)
        D.append(d) # μέχρι εδώ είναι όπως και προηγουμένως
    tmpd = D[0] # στο tmpd κρατάμε την τρέχουσα ελάχιστη απόσταση
    wherefrom = 0 # και στον ακέραιο wherefrom κρατάμε ένα σημείο της L όπου αυτή πιάνεται
    # αρχικές τιμές και των δύο είναι η απόσταση από το πρώτο στοιχείο της L και η θέση 0 (πρώτη θέση της
 L)
    for i in range(1,len(D)): # για κάθε σημείο της L από το 2ο και πέρα
        if D[i] < tmpd: # αν η απόστασή του από το q (έχει ήδη υπολογιστεί στο D[i]) είναι μικρότερη από
 την
                        # τρέχουσα τότε ενημερώνουμε την τρέχουσα απόσταση και τη θέση της (wherefrom)
            tmpd = D[i]; wherefrom = i
    return [tmpd, wherefrom] # επιστρέφουμε το ζεύγος τιμών που βρήκαμε

# Στο παρακάτω καλούμε τη συνάρτηση newmindistance με το σημείο [0.1, 300] και τη λίστα σημείων
# [ [-1000, -2000], [0.1, -200], [3, 4], [-10000, -3] ]. Στη μεταβλητή d αποθηκεύεται η ελάχιστη απόσταση
# που επιστρέφει η mindistance και στη μεταβλητή i το σημείο της L όπου αυτή πιάνεται.
[d, i] = newmindistance([0.1, 300], [ [-1000, -2000], [0.1, -200], [3, 4], [-10000, -3] ])

print("Η ελάχιστη απόσταση είναι {} από το σημείο L[{}]".format(d, i))
```

```
Η ελάχιστη απόσταση είναι 296.01420574019755 από το σημείο L[2]
```

Τοπικές (local) και καθολικές (global) μεταβλητές

Τα αποτελέσματα των `print` στο παρακάτω πρόγραμμα είναι τουλάχιστον περίεργα εκ πρώτης όψεως. Για να ξεκαθαρίσετε μερικά πράγματα σχετικά με τις τοπικές και τις καθολικές μεταβλητές σε μια συνάρτηση κι ένα πρόγραμμα διαβάστε το αντίστοιχο κομμάτι από τη σελίδα [http://fourier.math.uoc.gr/prog2/files/nb/feb24.html (http://fourier.math.uoc.gr/prog2/files/nb/feb24.html)] του Προγραμματισμού ΙΙ (Εαρινό Εξάμηνο 2014-15). Η σελίδα αυτή αφορά την Python 2 αλλά η μόνη διαφορά που θα δείτε εσείς είναι ότι το `print` συντάσσεται χωρίς παρενθέσεις (όπως στην Python 3).

In [11]:
```
y=5

def f(x):
    y = 10
    x = x+y
    print("In function: x=", x)
    return True

x = 1
f(x)
print("In main program: x=", x)
print("In main program: y=", y)
```

```
In function: x= 11
In main program: x= 1
In main program: y= 5
```