Σημειωματάριο μαθήματος 1ης Νοε. 2017

Παραδείγματα συναρτήσεων. Αναδρομικές συναρτήσεις. ¶

Ξεκινήσαμε πακετάροντας παλαιότερό μας κώδικα για τον υπολογισμό των διαιρετών ενός φυσικού αριθμού σε συνάρτηση.

Η συνάρτηση `divisors` παρακάτω επιστρέφει μια λίστα με όλους τους διαιρέτες του αριθμού n, του 1 και του n συμπεριλαμβανομένων. Δουλεύει πολύ απλά, διανύοντας όλους τους φυσικούς αριθμούς από το 2 έως το n-1 και ελέγχοντας αν διαιρούν το n. (Προσθέτει επίσης στη λίστα το 1 και το n που είναι κι αυτοί διαιρέτες του n.)

In [ ]:
```
def divisors(n):
    L = [1]
    for k in range(2, n):
        if n%k == 0:
            L.append(k)
    L.append(n)
    return L
```

Γράψαμε και την πιο γρήγορη έκδοση της συνάρτησης αυτής, που ελέγχει πιο έξυπνα μόνο όσους υποψήφιους διαιρέτες είναι $\leq \sqrt{n}$. Ονομάσαμε αυτή τη συνάρτηση `divisorsfast`.

In [ ]:
```
def divisorsfast(n):
    L = [1]
    for k in range(2, n):
        if k*k > n:
            break
        if n%k == 0:
            L.append(k)
            if k != n//k:
                L.append(n//k)
    L.append(n)
    return L
```

Η συνάρτηση `commondivisors(m, n)` χρησιμοποιεί τη συνάρτηση `divisorsfast` ή τη συνάρτηση `divisors` για να βρει τους κοινούς διαιρέτες των m και n. Τέλος η συνάρτηση `gcd(m, n)` χρησιμοποιεί τη συνάρτηση `commondivisors` για να βρει το μέγιστο κοινό διαιρέτη των m και n.

In [1]:
```
def divisors(n):
    L = [1]
    for k in range(2, n):
        if n%k == 0:
            L.append(k)
    L.append(n)
    return L

def divisorsfast(n):
    L = [1]
    for k in range(2, n):
        if k*k > n:
            break
        if n%k == 0:
            L.append(k)
            if k != n//k:
                L.append(n//k)
    L.append(n)
    return L

def commondivisors(m, n):
    Lm = divisorsfast(m)
    Ln = divisorsfast(n)
    L = []
    for d in Lm:
        if d in Ln:
            L.append(d)
    return L

def gcd(m, n):
    return max(commondivisors(m, n))

print(gcd(100, 60))
```

```
20
```

Η επόμενη συνάρτηση που γράψαμε είναι η συνάρτηση `factorial(n)` για τον υπολογισμού του παραγοντικού του n. Θυμίζουμε ότι

$$n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdots (n-1) \cdot n$$

με τη σύμβαση ότι $0! = 1$.

In [2]:
```
def factorial(n):
    product = 1
    for i in range(1, n+1):
        product *= i
    return product

print(factorial(5))
```

```
120
```

Γράψαμε έπειτα ξανά την ίδια συνάρτηση, με όνομα τώρα `factorialr` με τρόπο ώστε να εκμεταλλευτούμε την απλή ιδιότητα

$$n! = n(n-1)! \; \gamma\iota\alpha \; n > 0 \; .$$

Η συνάρτηση `factorialr` είναι ουσιαστικά μια μετάφραση του παραπάνω τύπου σε πρόγραμμα. Έτσι όταν ξεκινάει η συνάρτηση ελέγχει πρώτα απ' όλα αν είμαστε στην απλή περίπτωση $n = 0$ οπότε επιστρέφει 1. Αν όχι τότε η συνάρτηση καλεί τον εαυτό της για να υπολογίσει το $(n-1)!$, και αφού επιστρέψει αυτή η κλήση πολλαπλασιάζει με n και επιστρέφει το αποτέλεσμα. Όταν μια συνάρτηση καλεί τον εαυτό της λέμε ότι έχουμε μια αναδρομική (recursive) συνάρτηση.

Παρατηρείστε πόσο απλά είναι γραμμένη τώρα η συνάρτηση. Δώστε επίσης ιδιαίτερη σημασία στην περίπτωση $n = 0$, που είναι ακριβώς η περίπτωση όπου η συνάρτηση δεν καλεί τον εαυτό της, εκεί δηλ. όπου, όπως λέμε, σταματάει η αναδρομή. Αν κάποια στιγμή δεν σταματήσουν οι αναδρομικές κλήσεις τότε η συνάρτηση θα καλεί συνεχώς τον εαυτό της και δε θα τελειώσει ποτέ.

In [3]:
```
def factorialr(n):
    if n==0:
        return 1
    return n*factorialr(n-1)

print(factorialr(5))
```

```
120
```

Χρησιμοποιούμε τη συνάρτηση `factorial` (ή την `factorialr`) για να υπολογίσουμε τους *διωνυμικούς συντελεστές*

$$C(n,k) \ \gamma\iota\alpha \ 0 \leq k \leq n.$$

Ο ποσότητα αυτή ορίζεται να είναι το πόσα υποσύνολα μεγέθους k έχει το σύνολο $\{1,2,\ldots,n\}$ και αποδεικνύεται ότι ισχύει

$$C(n,k) = \frac{n!}{k!(n-k)!}.$$

Με αυτόν ακριβώς τον τύπο υλοποιούμε τη συνάρτηση `binomial(n, k)`.

In [4]:
```
def factorial(n):
    product = 1
    for i in range(1, n+1):
        product *= i
    return product

def binomial(n, k):
    if not (0 <= k <= n):
        return 0
    return factorial(n)//(factorial(k)*factorial(n-k))

print(binomial(10, 4))
```

```
210
```

Το λεγόμενο *τρίγωνο του Pascal* δεν είναι τίποτε άλλο από την ταυτότητα

$$C(n,k) = C(n-1,k-1) + C(n-1,k)$$

μέσω της οποίας γράφουμε τώρα μια αναδρομική μορφή της συνάρτησης `binomial`, που την ονομάζουμε `binomialr`, και η οποία δε χρησιμοποιεί τη συνάρτηση `factorial`.

Η συνθήκη με την οποία κόβουμε την αναδρομή είναι η περίπτωση που το k είναι 0 ή n οπότε ξέρουμε ότι η απάντηση είναι 1.

In [5]:
```
def binomialr(n, k):
    if k in [0, n]:
        return 1
    return binomialr(n-1, k-1) + binomialr(n-1, k)

print(binomialr(5, 3))
```

```
10
```

Η ακολουθία Fibonacci είναι η ακολουθία F_n, για $n \geq 0$, που ορίζεται ως εξής:

$$F_0 = F_1 = 1$$

και

$$F_n = F_{n-1} + F_{n-2} \ \ \gamma\iota\alpha\, n \geq 2.$$

Σε αυτή την ακολουθία η αναδρομή είναι ουσιαστικά ενσωματωμένη στον ορισμό της και η υλοποίησή της είναι σχεδόν μια μεταγραφή από την κοινή γλώσσα σε python.

In [6]:
```
def fib(n):
    if n<=1:
        return 1
    return fib(n-1)+fib(n-2)

print(fib(5))
```

```
8
```

Το δυαδικό ανάπτυγμα ενός φυσικού αριθμού

Αν b είναι ένας φυσικός αριθμός τότε συμβολίζουμε με

$$(x_n x_{n-1} \cdots x_2 x_1 x_0)_b$$

τον αριθμό (εδώ ισχύει $x_0, x_1, \ldots, x_n \in \{0, 1, 2, \ldots, b-1\}$)

$$x = x_0 + x_1 b + x_2 b^2 + x_3 b^3 + \cdots x_{n-1} b^{n-1} + x_n b^n.$$

Η έκφραση $(x_n x_{n-1} \cdots x_2 x_1 x_0)_b$ ονομάζεται το b-αδικό ανάπτυγμα του x και τα x_j ονομάζονται b-αδικά ψηφία του x. Η συνηθισμένη περίπτωση είναι η $b = 10$ οπότε μιλάμε για το 10-αδικό ανάπτυγμα του x, και τα x_j είναι τα συνηθισμένα του δεκαδικά ψηφία, που είναι όλα από 0 έως 9. Δεν υπάρχει όμως κάτι πραγματικό ιδιαίτερο με την περίπτωση $b = 10$ και κάθε αριθμός έχει ένα (μοναδικό) ανάπτυγμα σε κάθε βάση b. Μια ιδιαίτερα σημαντική περίπτωση είναι η περίπτωση $b = 2$. Στο δυαδικό ανάπτυγμα του x

$$x = x_0 + x_1 2 + x_2 4 + x_3 2^3 + \cdots x_{n-1} 2^{n-1} + x_n 2^n.$$

όλα τα ψηφία x_j είναι είτε 0 είτε 1. Στην παρακάτω συνάρτηση `binary(n)` υπολογίζουμε τα δυαδικά ψηφία του `n` σε ένα string. Το τελευταίο γράμμα του string είναι το ψηφίο των μονάδων (το x_0). Από την παραπάνω εξίσωση για το x προκύπτει ότι ο αριθμός $x - x_0$ είναι άρτιος και επίσης ότι

$$\frac{x - x_0}{2} = x_1 + x_2 2 + x_3 2^2 + \cdots + x_{n-1} 2^{n-2} + x_n 2^{n-1}.$$

Η τελευταία ισότητα μας λέει ότι το δυαδικό ανάπτυγμα του $\frac{x-x_0}{2}$ είναι το $(x_n x_{n-1} \cdots x_2 x_1)_2$. Σε αυτή την παρατήρηση στηρίζουμε την αναδρομική κλήση. Η άλλη βασική παρατήρηση είναι ότι το x_0 είναι 0 αν το x είναι άρτιο και 1 αν το x είναι περιττό.

In [7]:
```
def binary(n):
    if n==0:
        return "0"
    if n==1:
        return "1"
    if n%2 == 0:
        return binary(n//2)+"0"
    else:
        return binary((n-1)//2)+"1"

print(binary(1352))
```

```
10101001000
```

Ο αλγόριθμος του Ευκλείδη για το ΜΚΔ

Ο προηγούμενος αλγόριθμος για το μέγιστο κοινό διαιρέτη (ΜΚΔ) δύο αριθμών που υλοποιήσαμε σήμερα είναι πολύ απλός αλλά μπορεί επίσης να είναι και πολύ αργός. Ο λόγος είναι ότι στηρίζεται στην εύρεση όλων των διαιρετών των δύο αριθμών, και αυτή η διαδικασία είναι πάρα πολύ αργή όταν οι αριθμοί είναι μεγάλοι.

Ο αλγόριθμος του Ευκλείδη όμως, που είναι αυτός που πάντα χρησιμοποιούμε για την εύρεση του ΜΚΔ, είναι πολύ πιο γρήγορος και δε χρειάζεται να βρει τους διαιρέτες των δύο αριθμών. Η βασική παρατήρηση που κάνει τον αλγόριθμο να δουλεύει είναι η εξής.

Αν $a \geq b > 0$ είναι δύο φυσικοί αριθμοί και $r \in \{0, 1, 2, \ldots, b-1\}$ είναι το υπόλοιπο της διαίρεσης a/b τότε ο ΜΚΔ των a, b είναι ίσος με τον ΜΚΔ των b, r. Ο μεγαλύτερος δηλ. από τους δύο αριθμούς, ο a, αντικαταστάθηκε από έναν άλλο, τον r, ο οποίος είναι μικρότερος από τον μικρότερο από τους αρχικούς δύο αριθμούς, δηλ. τον b. Εφαρμόζοντας συνεχώς αυτή την αναγωγή φτάνουμε σε κάποια στιγμή σε ένα ζεύγος αριθμών όπου το υπόλοιπο είναι 0, όπου δηλ. ο μικρότερος από τους δύο αριθμούς διαιρεί το μεγαλύτερο. Σε αυτή την περίπτωση είναι προφανές ότι ο ΜΚΔ είναι ο μικρότερος από τους δύο αριθμούς.

Η συνάρτηση `gcdeuclid(a, b)` που βλέπετε παρακάτω υλοποιεί ακριβώς αυτό τον αλγόριθμο. Ελέγχουμε πρώτα αν ο χρήστης πέρασε στον αριθμό b κάποιον που είναι μικρότερος από τον a. Αν αυτό έγινε τότε ανταλλάσουμε τις τιμές των δύο ορισμάτων ώστε από δω και πέρα να είμαστε σίγουροι ότι ο a είναι ο μεγαλύτερος.

Έπειτα ελέγχουμε αν ο b διαιρεί τον a. Αν αυτό συμβαίνει τότε ο ΜΚΔ είναι φυσικά ο b, και άρα επιστρέφουμε αυτό.

Αν το υπόλοιπο `r` δεν είναι 0 τότε επιστρέφουμε το ΜΚΔ των b και `r`. Πώς τον βρίσκουμε αυτόν; Απλά η συνάρτηση `gcdeuclid` καλεί τον εαυτό της (με άλλα ορίσματα φυσικά από αυτά που της είχαμε περάσει). Είναι δηλ. αναδρομική (recursive) συνάρτηση.

In [8]:
```
def gcdeuclid(a, b):
    if a<b:
        a, b = b, a
    r = a%b
    if r == 0:
        return b
    return gcdeuclid(b, r)

print(gcdeuclid(1002342, 15132423))
```

```
3
```