Εργαστήριο 5

Εισαγωγή στον Προγραμματισμό Υπολογιστών

Περιεχόμενα

Αναδρομικές συναρτήσεις

1. GNU is Not Unix
2. Άθροισμα φυσικών αριθμών με αναδρομή
3. Άθροισμα ψηφίων με αναδρομή
4. Αναδρομική αντιστροφή συμβολοσειράς
5. Δέντρο από συμβολοσειρά

Παράδειγμα 1: GNU is Not Unix

Αναδρομικές συναρτήσεις

- GNU: οργανισμός ανάπτυξης ελεύθερου λογισμικού
  - Γνωστές εφαρμογές: Emacs, gcc (μεταφραστής γλώσσας C), bash, GNOME,…

GNU

- GNU = GNU is Not Unix

- Αναδρομικό όνομα: θα μπορούσαμε να αντικαθιστούμε το GNU στον ορισμό συνεχώς, χωρίς τέλος
  - GNU = GNU is Not Unix = GNU is Not Unix is Not Unix = GNU is Not Unix is Not Unix is Not Unix = …

Αναδρομικές συναρτήσεις

- Συμπληρώστε τα κενά με τη ίδια έκφραση χωρίς να περιέχει τη συμβολοσειρά `'GNU'`

```
>>> name = 'GNU'
>>> name = ____________________
>>> name
'GNU is Not Unix'
>>> name = ____________________
>>> name
'GNU is Not Unix is Not Unix'
>>> name = ____________________
>>> name = ____________________
>>> name = ____________________
>>> name
'GNU is Not Unix is Not Unix is Not Unix is Not Unix is Not Unix'
```

Αναδρομικές συναρτήσεις

- Συμπληρώστε τα κενά με τη ίδια έκφραση χωρίς να περιέχει τη συμβολοσειρά `'GNU'`

```
>>> name = 'GNU'
>>> name = name + ' is Not Unix'
>>> name
'GNU is not Unix'
>>> name = name + ' is Not Unix'
>>> name
'GNU is Not Unix is Not Unix'
>>> name = name + ' is Not Unix'
>>> name = name + ' is Not Unix'
>>> name = name + ' is Not Unix'
>>> name
'GNU is Not Unix is Not Unix is Not Unix is Not Unix is Not Unix'
```

Αναδρομικές συναρτήσεις

- Θα κατασκευάσουμε βήμα-βήμα *αναδρομική* συνάρτηση `name` με το ίδιο αποτέλεσμα:

```
>>> name(0)
'GNU'
>>> name(1)
'GNU is Not Unix'
>>> name(2)
'GNU is Not Unix is Not Unix'
>>> name(3)
'GNU is Not Unix is Not Unix is Not Unix'
```

Αναδρομικές συναρτήσεις

- **Βήμα 1:** κατασκευή συνάρτησης `name0(n)` που λειτουργεί σωστά μόνο για τη βασική περίπτωση `n == 0`

```
>>> def name0(n):
        ____________________

>>> name0(0)
'GNU'
```

Αναδρομικές συναρτήσεις

- **Βήμα 1:** κατασκευή συνάρτησης `name0(n)` που λειτουργεί σωστά μόνο για τη βασική περίπτωση `n == 0`

```
>>> def name0(n):
        return 'GNU'

>>> name0(0)
'GNU'
```

Αναδρομικές συναρτήσεις

- **Βήμα 2:** κατασκευή συνάρτησης `name1(n)` που λειτουργεί σωστά για `n == 1` και χρησιμοποιεί τη `name0(n-1)` εάν γίνεται:

```
>>> def name1(n):
        _____________________________________

>>> name1(1)
'GNU is Not Unix'
```

Αναδρομικές συναρτήσεις

- **Βήμα 2:** κατασκευή συνάρτησης `name1(n)` που λειτουργεί σωστά για `n == 1` και χρησιμοποιεί τη `name0(n-1)` εάν γίνεται:

```
>>> def name1(n):
        return name0(n-1) + ' is Not Unix'

>>> name1(1)
'GNU is Not Unix'
```

Εφόσον κατορθώσαμε να χρησιμοποιήσουμε τη `name0`, σημαίνει ότι έχουμε καταλάβει ποιο είναι το *αναδρομικό βήμα*.

Αναδρομικές συναρτήσεις

- **Βήμα 3:** κατασκευή αναδρομικής συνάρτησης `name(n)` που ενοποιεί τη λειτουργία των `name0` και `name1`:

```
>>> def name(n):
        if n == 0:
            ____________________
        else:
            ____________________

>>> name(0)
'GNU'
>>> name(1)
'GNU is Not Unix'
>>> name(2)
'GNU is Not Unix is Not Unix'
```

Αναδρομικές συναρτήσεις

- **Βήμα 3:** κατασκευή αναδρομικής συνάρτησης `name(n)` που ενοποιεί τη λειτουργία των `name0` και `name1`:

```
>>> def name(n):
        if n == 0:
            return 'GNU'
        else:
            return name(n-1) + ' is not Unix'

>>> name(0)
'GNU'
>>> name(1)
'GNU is Not Unix'
>>> name(2)
'GNU is Not Unix is Not Unix'
```

Αναδρομικές συναρτήσεις

- Για να ανακαλύψουμε το αναδρομικό βήμα, ακολουθήσαμε μια μέθοδο κατασκευής αναδρομικών συναρτήσεων σε βήματα:

1. Κατασκευάζουμε τη συνάρτηση `func1(n)` που λειτουργεί σωστά μόνο για τη βασική περίπτωση, πχ. για `n == 1`
2. Κατασκευάζουμε τη συνάρτηση `func2(n)` που λειτουργεί σωστά μόνο για την επόμενη περίπτωση, πχ. για `n == 2`, όπου προσπαθούμε να χρησιμοποιήσουμε κλήσεις στη `func1(n-1)`
3. Εάν καταφέραμε να χρησιμοποιήσουμε τη `func1(n-1)` τότε έχουμε καταλάβει ποιο είναι το αναδρομικό βήμα! Κατασκευάζουμε την αναδρομική συνάρτηση `func(n)` ως εξής:
   A. Το βασικό βήμα είναι το σώμα της συνάρτησης `func1`. Προσέξτε να προσθέσετε τον έλεγχο συνθήκης `n==1`
   B. Το αναδρομικό βήμα είναι αυτό που προκύπτει από το βήμα 2 παραπάνω. Αντικαθιστούμε όλες τις κλήσεις στη `func1(n-1)` με κλήσεις στη `func(n-1)`
4. Εάν δεν το καταφέραμε, προχωράμε στην κατασκευή της `func3(n)` που λειτουργεί σωστά μόνο για την περίπτωση `n == 3` και προσπαθούμε να χρησιμοποιήσουμε κλήσεις στη `func2(n-1)` και συνεχίζουμε όπως παραπάνω…

Παράδειγμα 2: αναδρομική άθροιση φυσικών αριθμών

Αναδρομικές συναρτήσεις

- Θα κατασκευάσουμε την *αναδρομική* συνάρτηση `sum_naturals(n)` η οποία επιστρέφει το άθροισμα των φυσικών αριθμών 1 έως και `n`

```
>>> sum_naturals(3)
6
>>> sum_naturals(4)
10
>>> sum_naturals(100)
5050
```

Αναδρομικές συναρτήσεις

- **Βήμα 1:** κατασκευή συνάρτησης `sum_naturals1(n)` που λειτουργεί σωστά μόνο για τη βασική περίπτωση `n==1`

```
>>> def sum_naturals1(n):
        ____________________

>>> sum_naturals1(1)
1
```

Αναδρομικές συναρτήσεις

- **Βήμα 1:** κατασκευή συνάρτησης `sum_naturals1(n)` που λειτουργεί σωστά μόνο για τη βασική περίπτωση `n==1`

```
>>> def sum_naturals1(n):
        return 1

>>> sum_naturals1(1)
1
```

Αναδρομικές συναρτήσεις

- **Βήμα 2:** κατασκευή συνάρτησης `sum_naturals2(n)` που δίνει το σωστό αποτέλεσμα για `n==2`

```
>>> def sum_naturals2(n):
        _____________________________________

>>> sum_naturals2(2)
3
```

Προσπαθήστε να χρησιμοποιήσετε κλήση στη `sum_naturals1`

Χρησιμοποιήστε `n`, `n-1` αντί των αριθμών `2`, `1`

Αναδρομικές συναρτήσεις

- **Βήμα 2:** κατασκευή συνάρτησης `sum_naturals2(n)` που δίνει το σωστό αποτέλεσμα για `n==2`

```
>>> def sum_naturals2(n):
        return sum_naturals1(n - 1) + n


>>> sum_naturals2(2)
3
```

Προσπαθήστε να χρησιμοποιήσετε κλήση στη `sum_naturals1`

Χρησιμοποιήστε `n`, `n-1` αντί των αριθμών `2`, `1`

Αναδρομικές συναρτήσεις

- **Βήμα 3:** κατασκευή τελικής συνάρτησης `sum_naturals(n)` **ενοποιώντας τις** `sum_naturals1, sum_naturals2`

```
>>> def sum_naturals(n):
        _____________________________________
        _____________________________________
        _____________________________________
        _____________________________________

>>> sum_naturals(1)
1
>>> sum_naturals(3)
6
```

Αναδρομικές συναρτήσεις

- **Βήμα 3:** κατασκευή τελικής συνάρτησης `sum_naturals(n)` **ενοποιώντας τις** `sum_naturals1, sum_naturals2`

```
>>> def sum_naturals(n):
        if n == 1:
            return 1
        else:
            return sum_naturals(n - 1) + n

>>> sum_naturals(1)
1
>>> sum_naturals(3)
6
```

Παράδειγμα 3: αναδρομική άθροιση ψηφίων

Αναδρομικές συναρτήσεις

- Θα κατασκευάσουμε την *αναδρομική* συνάρτηση `sum_digits(x)` η οποία επιστρέφει το άθροισμα των ψηφίων του `x`

```
>>> sum_digits(123)
6
>>> sum_digits(100033)
7
```

Αναδρομικές συναρτήσεις

- **Βήμα 1:** κατασκευή συνάρτησης `sum_digits1(x)` που λειτουργεί σωστά μόνο για τη βασική περίπτωση όπου το `x` είναι μονοψήφιος

```
>>> def sum_digits1(x):
        ____________________

>>> sum_digits1(0)
0
>>> sum_digits1(9)
9
```

Αναδρομικές συναρτήσεις

- **Βήμα 1:** κατασκευή συνάρτησης `sum_digits1(x)` που λειτουργεί σωστά μόνο για τη βασική περίπτωση όπου το `x` είναι μονοψήφιος

```
>>> def sum_digits1(x):
        return x

>>> sum_digits1(0)
0
>>> sum_digits1(9)
9
```

Αναδρομικές συναρτήσεις

- **Βήμα 2:** κατασκευή συνάρτησης `sum_digits2(x)` που δίνει το σωστό αποτέλεσμα για διψήφιους `x`

```
>>> def sum_digits2(x):
        _____________________________________
        _____________________________________
        _____________________________________

>>> sum_digits2(23)
5
>>> sum_digits2(99)
18
```

Αναδρομικές συναρτήσεις

- **Βήμα 2:** κατασκευή συνάρτησης `sum_digits2(x)` που δίνει το σωστό αποτέλεσμα για διψήφιους `x`

```
>>> def sum_digits2(x):
        return (x // 10) + (x % 10)
```

```
>>> sum_digits2(23)
5
>>> sum_digits2(99)
18
```

Αναδρομικές συναρτήσεις

- **Βήμα 3:** κατασκευή συνάρτησης `sum_digits3(x)` που δίνει το σωστό αποτέλεσμα για τριψήφιους `x`

```
>>> def sum_digits3(x):
        _____________________________
        _____________________________
        _____________________________

>>> sum_digits3(123)
6
>>> sum_digits3(901)
10
```

Αναδρομικές συναρτήσεις

- **Βήμα 3:** κατασκευή συνάρτησης `sum_digits3(x)` που δίνει το σωστό αποτέλεσμα για τριψήφιους `x`

```
>>> def sum_digits3(x):
        return sum_digits2(x // 10) + x % 10
```

```
>>> sum_digits3(123)
6
>>> sum_digits3(901)
10
```

Αναδρομικές συναρτήσεις

- **Βήμα 3:** κατασκευή συνάρτησης `sum_digits3(x)` που δίνει το σωστό αποτέλεσμα για τριψήφιους `x`

```
>>> def sum_digits3(x):
        return sum_digits2(x // 10) + x % 10
```

αναδρομικό βήμα

```
>>> sum_digits3(123)
6
>>> sum_digits3(901)
10
```

Αναδρομικές συναρτήσεις

- **Βήμα 4:** κατασκευή τελικής συνάρτησης `sum_digits(x)` ενοποιώντας τις `sum_digits1`, `sum_digits2`, `sum_digits3`

```
>>> def sum_digits(x):
        if x <= 9:
            ____________________
        elif 10 <= x <= 99:
            ____________________
        else:
            ____________________

>>> sum_digits(2018)
11
```

Αν επιθυμείτε, μπορείτε να μη χρησιμοποιήσετε το `elif`

Αναδρομικές συναρτήσεις

- **Βήμα 4:** κατασκευή τελικής συνάρτησης `sum_digits(x)` ενοποιώντας τις `sum_digits1`, `sum_digits2`, `sum_digits3`

```
>>> def sum_digits(x):
        if x <= 9:
            return x
        elif 10 <= x <= 99:
            return x // 10 + x % 10
        else:
            return sum_digits(x // 10) + x % 10

>>> sum_digits(2018)
11
```

Αναδρομικές συναρτήσεις

- **Βήμα 4:** κατασκευή τελικής συνάρτησης `sum_digits(x)` ενοποιώντας τις `sum_digits1`, `sum_digits2`, `sum_digits3`

```
>>> def sum_digits(x):
        if x <= 9:
            return x
        else:
            return sum_digits(x // 10) + x % 10

>>> sum_digits(2018)
11
```

Παράδειγμα 4: αναδρομική αντιστροφή συμβολοσειράς

Αναδρομικές συναρτήσεις

- Θα χρησιμοποιήσουμε τις συναρτήσεις `first` και `rest` χειρισμού string που βρίσκονται στο module `string_manipulation`

- Κατεβάστε το `string_manipulation.py` από το eclass -> Έγγραφα -> Εργαστήριο 5

- Αντιγράψτε/μετακινήστε το σε κατάλογο από όπου μπορεί να γίνει `import` από το διαδραστικό περιβάλλον ή το πρόγραμμά σας

```
>>> from string_manipulation import first, rest
>>> first('hello') # επιστρέφει τον πρώτο χαρακτήρα ενός string
'h'
>>> rest('hello') # επιστρέφει το υπόλοιπο string
'ello'
```

Αναδρομικές συναρτήσεις

- Θα κατασκευάσουμε την *αναδρομική* συνάρτηση `reverse` η οποία επιστρέφει το αντεστραμμένο string

```
>>> reverse('abc')
'cba'
>>> reverse('hello')
'olleh'
>>> reverse('tit 4 tat')
'tat 4 tit'
```

Σημείωση: η αντιστροφή του string `s` συνήθως γίνεται με την έκφραση `s[::-1]`, που θα μιλήσουμε αργότερα στο μάθημα

Σκοπός εδώ είναι να την υλοποιήσουμε αναδρομικά και τη χρήση των `first`, `rest`

Αναδρομικές συναρτήσεις

- **Βήμα 1:** κατασκευή συνάρτησης `reverse1(s)` που λειτουργεί σωστά μόνο για τη βασική περίπτωση όπου το `s` αποτελείται από ένα χαρακτήρα

```
>>> reverse1('a')
'a'
```

- **Βήμα 2:** κατασκευή συνάρτησης `reverse2(s)` που λειτουργεί σωστά για το επόμενο βήμα (string 2 χαρακτήρων)

```
>>> reverse2('ab')
'ba'
```

- **Βήμα 3:** κατασκευή συνάρτησης `reverse3(s)` που λειτουργεί σωστά για το επόμενο βήμα (string 3 χαρακτήρων)

```
>>> reverse3('abc')
'cba'
```

- **Βήμα 4:** κατασκευή τελικής συνάρτησης `reverse(s)` ενοποιώντας τις `reverse1`, `reverse2`, `reverse3`

```
>>> reverse('abracadabra')
'arbadacarba'
```

Στη `reverse`, μπορείτε να χρησιμοποιήσετε τη συνάρτηση `len(s)` που επιστρέφει το πλήθος χαρακτήρων του `s`

Αναδρομικές συναρτήσεις

- **Βήματα 1-3:**

```
>>> def reverse1(s):
        return s

>>> def reverse2(s):
        return rest(s) + first(s)

>>> def reverse3(s):
        return reverse2(rest(s)) + first(s)
```

αναδρομικό βήμα

Αναδρομικές συναρτήσεις

- **Βήμα 4:** κατασκευή τελικής συνάρτησης `reverse(s)` **ενοποιώντας τις** `reverse1`, `reverse2`, `reverse3`

```
>>> def reverse(s):
        if len(s) == 1:
            return s
        elif len(s) == 2:
            return rest(s) + first(s)
        else:
            return reverse(rest(s)) + first(s)

>>> reverse('abracadabra')
'arbadacarba'
```

Αναδρομικές συναρτήσεις

- **Βήμα 4:** κατασκευή τελικής συνάρτησης `reverse(s)` **ενοποιώντας τις** `reverse1`, `reverse2`, `reverse3`

```
>>> def reverse(s):
        if len(s) == 1:
            return s
        else:
            return reverse(rest(s)) + first(s)

>>> reverse('abracadabra')
'arbadacarba'
```

Αναδρομικές συναρτήσεις

- Υλοποιήστε τη συνάρτηση `reverse` χρησιμοποιώντας αναδρομή στην τελική κλήση (*tail recursion*)

```
>>> def reverse(_________________):
        if len(s) == 0:
            return ___________________
        else:
            return reverse(_________________________)

>>> reverse('abracadabra')
'arbadacarba'
```

Αναδρομικές συναρτήσεις

- Υλοποιήστε τη συνάρτηση `reverse` χρησιμοποιώντας αναδρομή στην τελική κλήση (*tail recursion*)

```
>>> def reverse(s, result = ''):
        if len(s) == 0:
            return result
        else:
            return reverse(rest(s), first(s) + result)

>>> reverse('abracadabra')
'arbadacarba'
```

Παράδειγμα 5: Δέντρο από συμβολοσειρά

Αναδρομικές συναρτήσεις

- Κατασκευάστε *αναδρομική* συνάρτηση `string_tree` που λειτουργεί όπως στο παρακάτω παράδειγμα
  - Μπορείτε να χρησιμοποιήσετε τη συνάρτηση `rest` από το module `string_manipulation`

```
>>> string_tree('hello')
hello
ello
llo
lo
o
```

Αναδρομικές συναρτήσεις

- Λύση:

```
def string_tree(s):
    if len(s) == 1:
        print(s)
    else:
        print(s)
        string_tree(rest(s))
```