# Εισαγωγή στον Προγραμματισμό Υπολογιστών

Διάλεξη 5

Αναδρομή και Αναδρομικοί Υπολογισμοί

# Περιεχόμενα

1. Αναδρομικές συναρτήσεις
2. Αναδρομή στην τελική κλήση μόνο (tail recursion)
3. Αμοιβαία αναδρομή
4. Δενδροειδής αναδρομή
5. Παράδειγμα: ο πύργος του Ανόι

# Αναδρομικές συναρτήσεις

```
>>> repeat(10, '*')
'**********'
```

```
repeat(9, '*') + '*'
```

# Αναδρομικές συναρτήσεις

```
>>> repeat(10, '*')
'**********'

>>> def repeat(n, symbol):
        """Repeats symbol."""
        if n == 1:
            return symbol
        else:
            return repeat(n-1, symbol) + symbol
```

# Αναδρομικές συναρτήσεις

```
1  def repeat(n, symbol):
2      """Repeats symbol."""
3      if n == 1:
4          return symbol
5      else:
6          return repeat(n-1, symbol) + symbol
7
8  res = repeat(5, '*')
```

μετάβαση στο Pythontutor

# Αναδρομικές συναρτήσεις

- **Αναδρομική συνάρτηση**: συνάρτηση η οποία στο σώμα της περιέχει (άμεσα ή έμμεσα) κλήσεις στην ίδια συνάρτηση
- Οι κλήσεις αφορούν μικρότερα στιγμιότυπα του ίδιου υπολογισμού
- Ειδικός χειρισμός του μικρότερου στιγμιότυπου (βασική περίπτωση)

# Αναδρομικές συναρτήσεις

- Παραγοντική συνάρτηση:

$$n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n = (n-1)! \cdot n$$

$$0! = 1$$

```
def fact(n):
    """Factorial."""
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```

# Αναδρομικές συναρτήσεις

```
1  def fact(n):
2      """Factorial."""
3      if n == 0:
4          return 1
5      else:
6          return n * fact(n - 1)
7
8  fact(4)
```

μετάβαση στο Pythontutor

# Αναδρομικές συναρτήσεις

- Σύγκριση με επαναληπτικό υπολογισμό

```
1  def fact_iter(n):
2      """Factorial."""
3      i, total = 1, 1
4      while i <= n:
5          total, i = total*i, i+1
6      return total
7
8  fact_iter(4)
```

μετάβαση στο Pythontutor

# Αναδρομικές συναρτήσεις

- Εμφάνιση δέντρου με *

```
>>> print_tree(10)
*
**      print_tree(9)
***
****
*****
******
*******
********
*********
**********
```

```
def print_tree(n):
    """Prints a * tree."""
    if n == 0:
        return
    else:
        print_tree(n-1)
        print(n*'*')
```

# Αναδρομικές συναρτήσεις

- Εμφάνιση δέντρου με *

```
>>> print_tree(10)
**********
*********
********
*******
******
*****
****
***
**
*
```

```
def print_tree(n):
    """Prints a * tree."""
    if n == 0:
        return
    else:
        print(n*'*')
        print_tree(n-1)
```

# Αναδρομικές συναρτήσεις

- Εμφάνιση δέντρου με *

```
>>> print_tree(5)
*****
****
***
**
*
*
**
***
****
*****
```

```
def print_tree(n):
    """Prints a * tree."""
    if n == 0:
        return
    else:
        print(n*'*')
        print_tree(n-1)
        print(n*'*')
```

# Αναδρομή στην τελική κλήση μόνο (*tail recursion*)

# Tail recursion

- **Αναδρομή στην τελική κλήση (*tail recursion*):** η αναδρομική κλήση γίνεται μόνο στην τελευταία εντολή που εκτελείται μέσα στη συνάρτηση
  - Πχ,

```
def print_tree(n):
    """Prints a * tree."""
    if n == 0:
        return
    else:
        print(n*'*')
        print_tree(n-1)
```

  - Οι τοπικές μεταβλητές δεν θα αποτιμηθούν ξανά
  - Σε ορισμένες γλώσσες (tail recursive) η κλήση `print_tree(n-1)` χρησιμοποιεί το ίδιο πλαίσιο με την `print_tree(n)`, χωρίς να δημιουργηθεί νέο
    - Γρηγορότερη εκτέλεση με εξοικονόμηση μνήμης

# Tail recursion

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

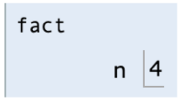
```
def fact(n):
    """Factorial."""
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```

```
def fact_tail(n, total=1):
    """Factorial."""
    if n == 0:
        return total
    else:
        return fact_tail(n - 1, n * total)
```

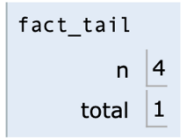
# Tail recursion

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

```
fact(4)
```



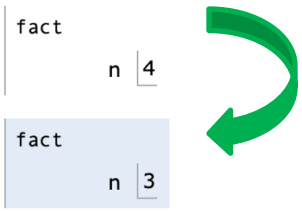

```
fact_tail(4)
```



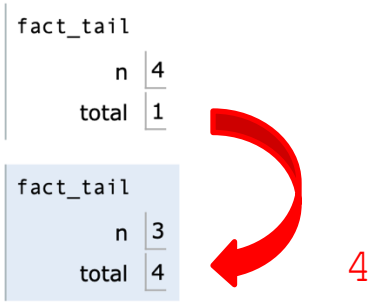
# Tail recursion

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

`fact(4)`




`fact_tail(4)`

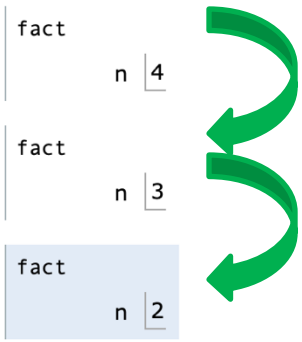



4

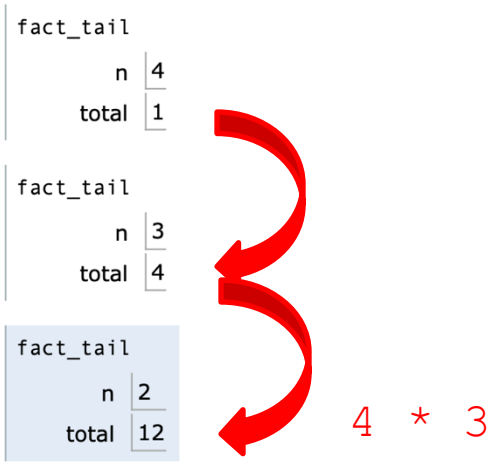
# Tail recursion

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

`fact(4)`




`fact_tail(4)`

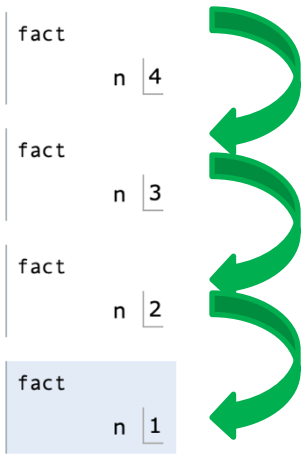



4 * 3

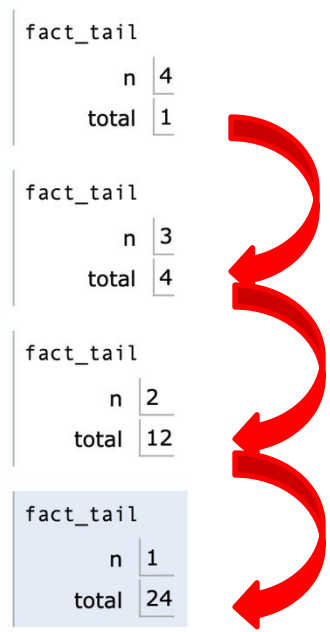
# Tail recursion

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

`fact(4)`




`fact_tail(4)`

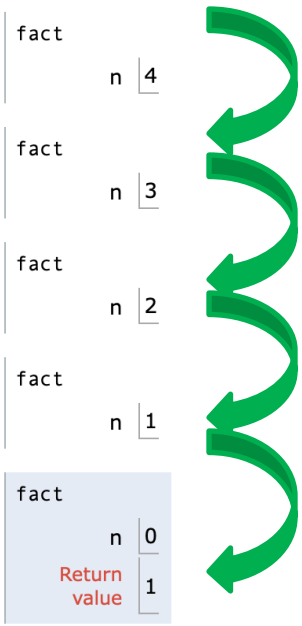



4 * 3 * 2

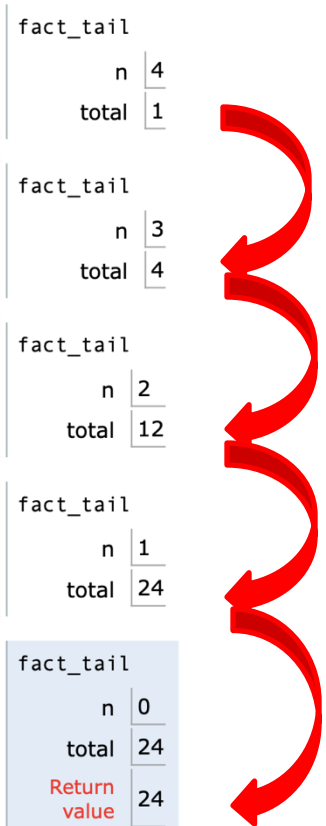
# Tail recursion

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

`fact(4)`




`fact_tail(4)`

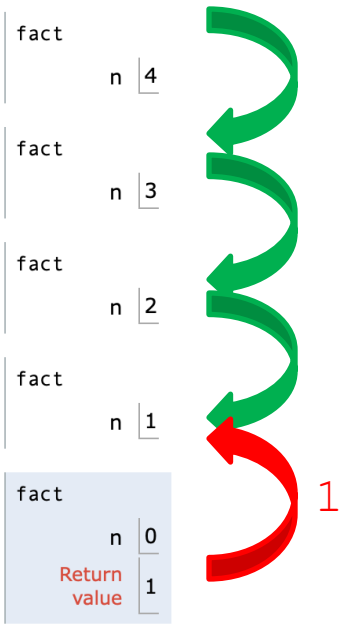



4 * 3 * 2 * 1

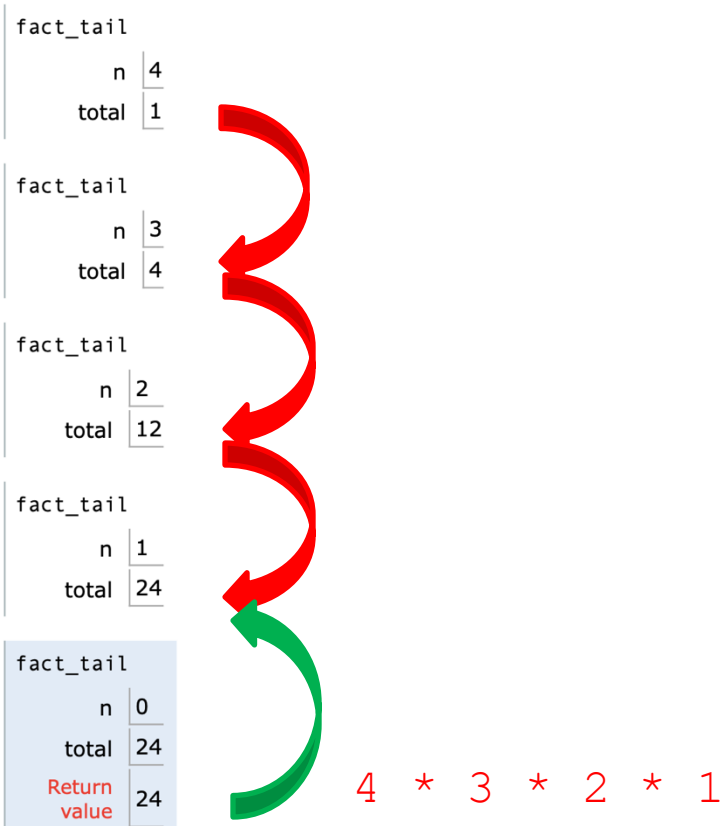
# Tail recursion

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

`fact(4)`

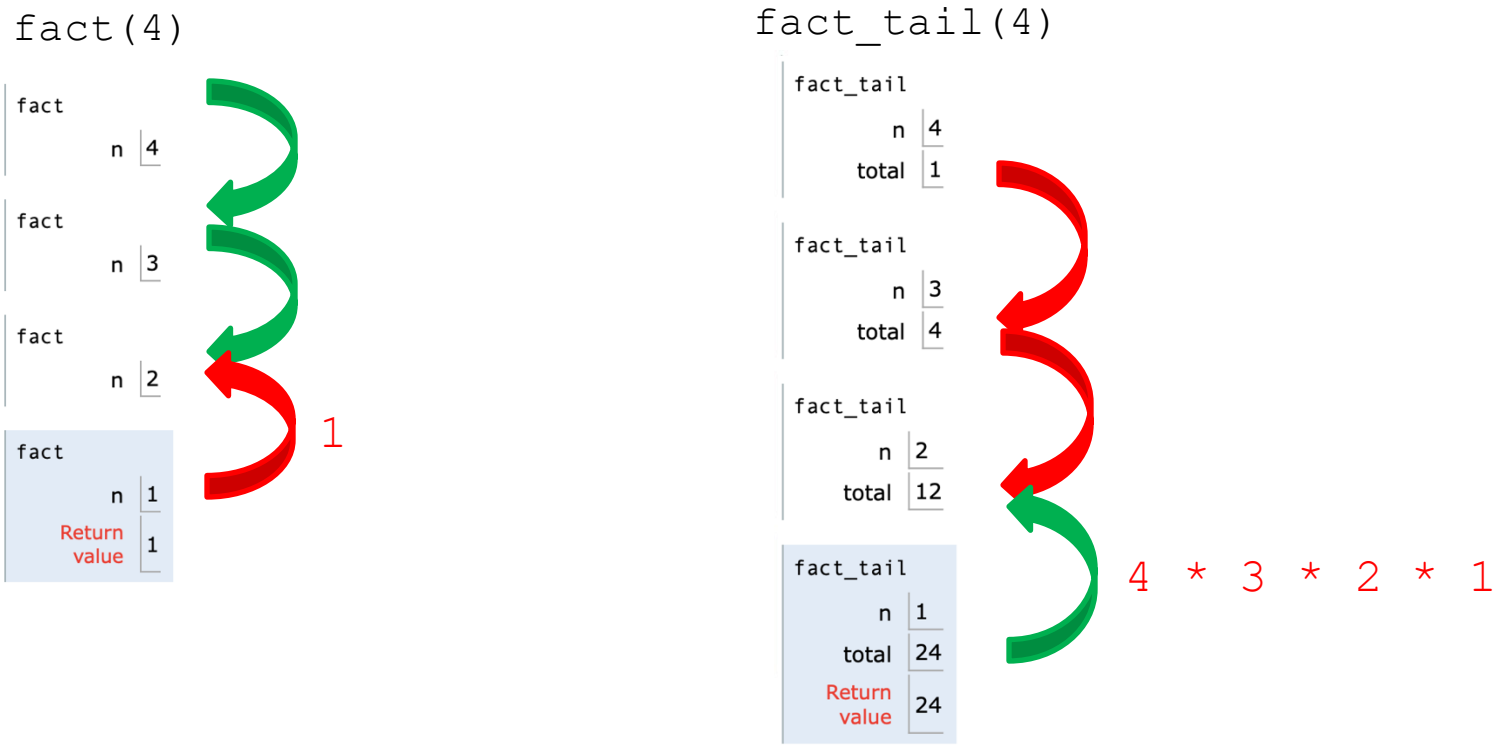



`fact_tail(4)`



# Tail recursion

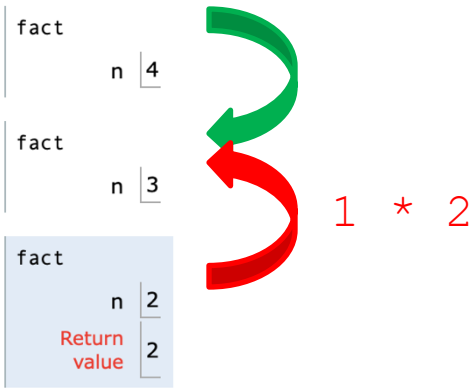
- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,



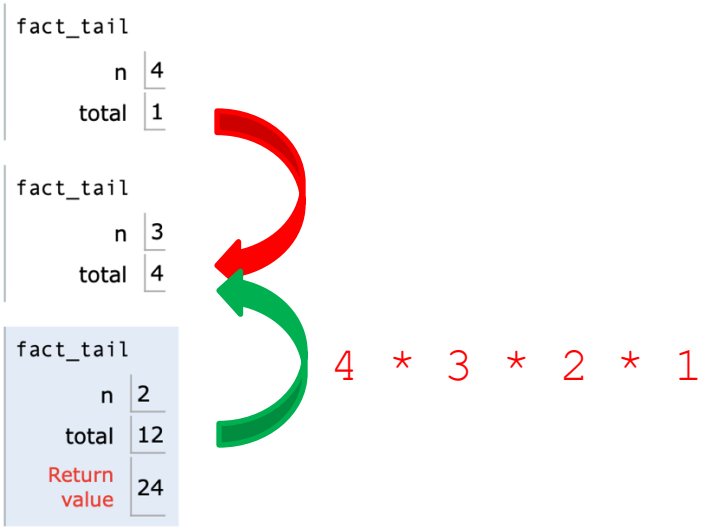
# Tail recursion

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

`fact(4)`



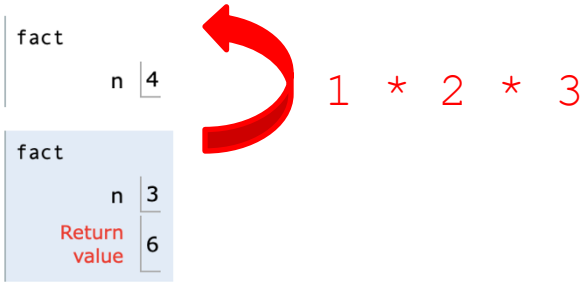

`fact_tail(4)`



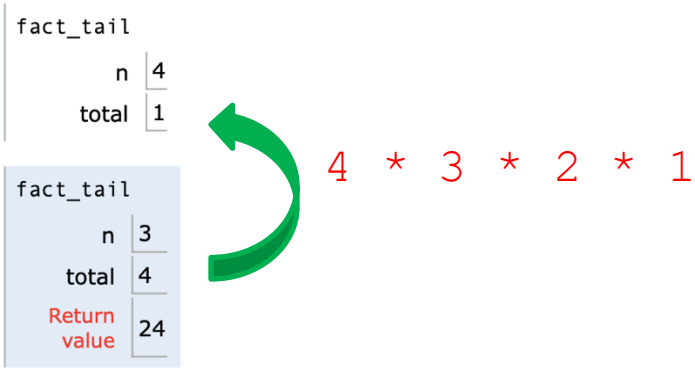
# Tail recursion

- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

`fact(4)`



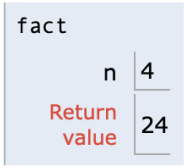

`fact_tail(4)`



# Tail recursion

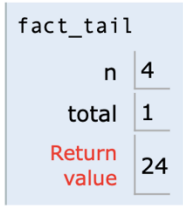
- Κάθε αναδρομική συνάρτηση μπορεί να υλοποιηθεί ισοδύναμα με tail recursion
  - Πχ,

`fact(4)`




`1 * 2 * 3 * 4`

`fact_tail(4)`




`4 * 3 * 2 * 1`

# Αναδρομικές συναρτήσεις

- Κάθε αναδρομικός υπολογισμός μπορεί να ξαναγραφεί ισοδύναμα με επαναληπτικό υπολογισμό (εντολές επανάληψης)
- Κάθε επαναληπτικός υπολογισμός μπορεί να ξαναγραφεί ισοδύναμα με αναδρομικό υπολογισμό

- Αναδρομικοί υπολογισμοί
  + ευκολότερος έλεγχος ορθότητας
  – σπατάλη μνήμης (στις μη tail-recursive γλώσσες)
- Επαναληπτικοί υπολογισμοί
  – δυσκολότερος έλεγχος ορθότητας
  + μικρότερη μνήμη

- Χρήση υπολογισμού καταλληλότερου για συγκεκριμένη περίπτωση

# Αμοιβαία αναδρομή

# Αμοιβαία αναδρομή

- **Αμοιβαία αναδρομή:** αναδρομικός υπολογισμός όπου μια συνάρτηση καλεί εμμέσως τον εαυτό της μέσω μιας άλλης

- Παράδειγμα: αναδρομικός ορισμός *άρτιων* αριθμών
  - Ο `n` είναι *άρτιος* εάν ο `n-1` είναι *περιττός*
  - Ο `n` είναι *περιττός* εάν ο `n-1` είναι *άρτιος*
  - Το `0` είναι *άρτιος*

# Αμοιβαία αναδρομή

- **Αμοιβαία αναδρομή:** αναδρομικός υπολογισμός όπου μια συνάρτηση καλεί εμμέσως τον εαυτό της μέσω μιας άλλης

```
def is_even(n):
    if n == 0:
        return True
    else:
        return is_odd(n-1)

def is_odd(n):
    if n == 0:
        return False
    else:
        return is_even(n-1)
```

# Αμοιβαία αναδρομή

- Αμοιβαία αναδρομικές συναρτήσεις μπορούν να μετατραπούν σε μια αναδρομική συνάρτηση αντικαθιστώντας την κλήση της μιας συνάρτησης με το σώμα της

```
def is_even2(n):
    if n == 0:
        return True
    elif n == 1:
        return False
    else:
        return is_even2(n-2)
```

# Δενδροειδής αναδρομή

# Δενδροειδής αναδρομή

- **Δενδροειδής αναδρομή:** περισσότερες από μια κλήσεις της αναδρομικής συνάρτησης με διαφορετικά ορίσματα

- Παράδειγμα: ακολουθία Fibonacci: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$

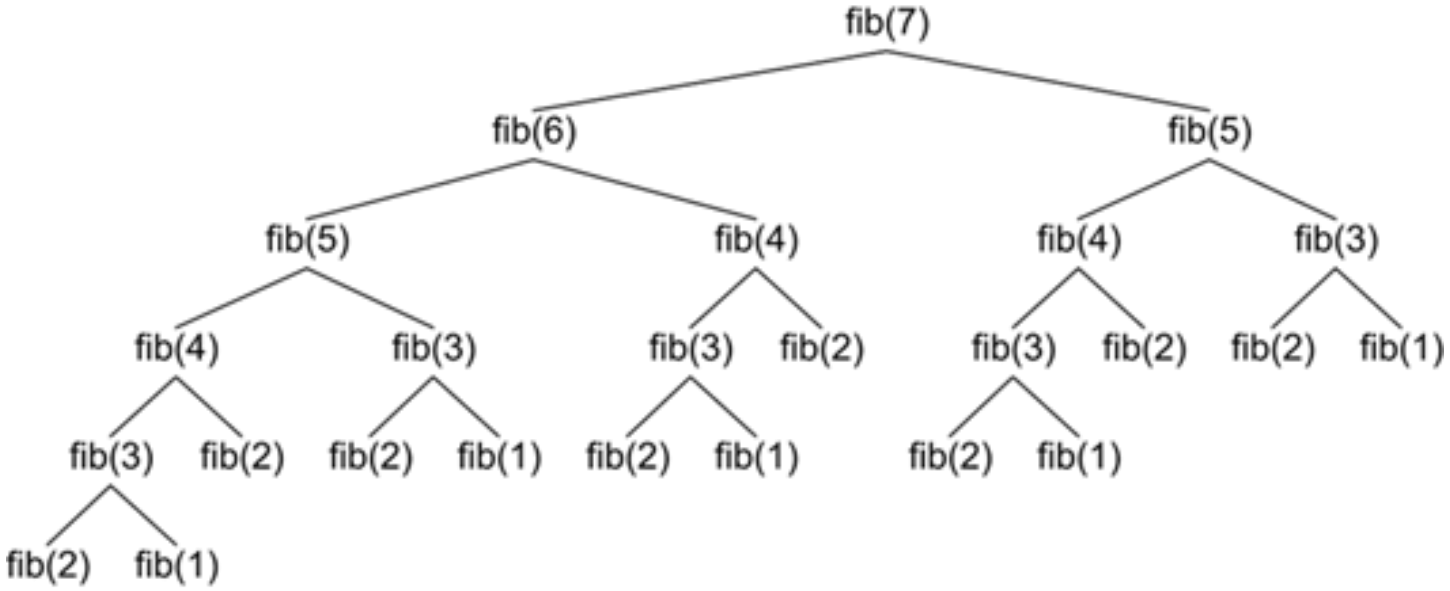
  $x_n = x_{n-1} + x_{n-2}$, εάν $n \geq 3$

  $x_1 = 0, x_2 = 1$

```
>>> def fib(n):
        if n >= 3:
            return fib(n-1) + fib(n-2)
        elif n == 2:
            return 1
        elif n == 1:
            return 0
```

# Δενδροειδής αναδρομή

- Οι κλήσεις της `fib` σχηματίζουν δέντρο




Εικόνα: http://www.introprogramming.info/english-intro-csharp-book/read-online/chapter-10-recursion/

# Παράδειγμα: ο πύργος του Ανόι

- Πως θα μετακινήσουμε τον πύργο χωρίς να τοποθετήσουμε μεγαλύτερο δίσκο πάνω σε μικρότερο;

By André Karwath aka Aka - Own work, CC BY-SA 2.5,
https://commons.wikimedia.org/w/index.php?curid=85401

# Παράδειγμα: ο πύργος του Ανόι

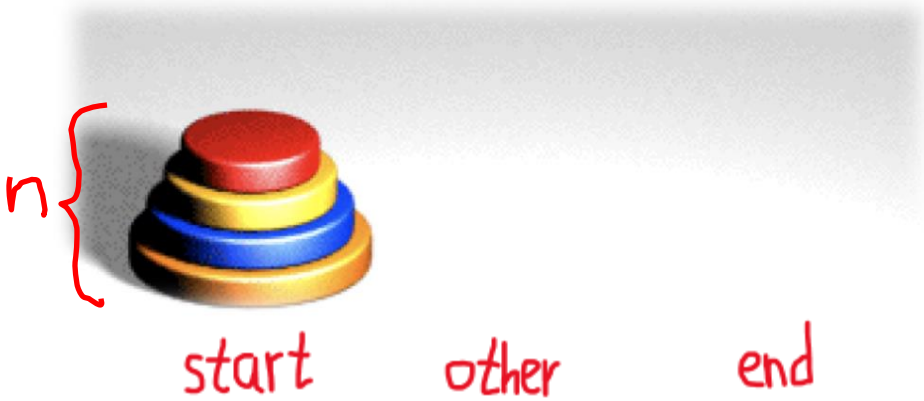
- Πως θα μετακινήσουμε τον πύργο χωρίς να τοποθετήσουμε μεγαλύτερο δίσκο πάνω σε μικρότερο;

By André Karwath aka Aka - Own work, CC BY-SA 2.5,
https://commons.wikimedia.org/w/index.php?curid=85401

# Παράδειγμα: ο πύργος του Ανόι

- Πως θα μετακινήσουμε τον πύργο χωρίς να τοποθετήσουμε μεγαλύτερο δίσκο πάνω σε μικρότερο;

```
def hanoi(n, start, end):
    if n == 1:
        print(start, end)
```

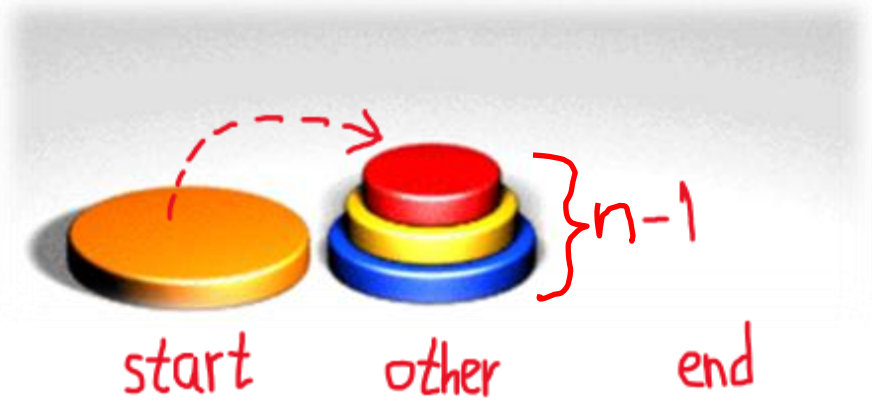



By André Karwath aka Aka - Own work, CC BY-SA 2.5,
https://commons.wikimedia.org/w/index.php?curid=85401

# Παράδειγμα: ο πύργος του Ανόι

- Πως θα μετακινήσουμε τον πύργο χωρίς να τοποθετήσουμε μεγαλύτερο δίσκο πάνω σε μικρότερο;

```
def hanoi(n, start, end):
    if n == 1:
        print(start, end)
    else:
        other = 6 - start - end
        hanoi(n-1, start, other)
```

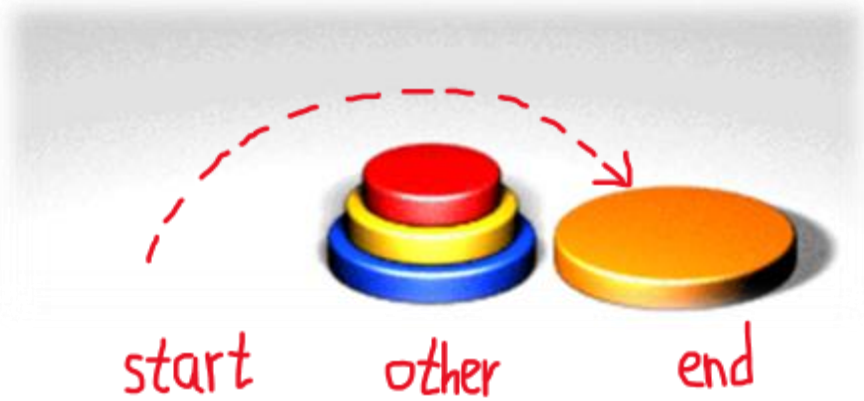



By André Karwath aka Aka - Own work, CC BY-SA 2.5,
https://commons.wikimedia.org/w/index.php?curid=85401

# Παράδειγμα: ο πύργος του Ανόι

- Πως θα μετακινήσουμε τον πύργο χωρίς να τοποθετήσουμε μεγαλύτερο δίσκο πάνω σε μικρότερο;

```
def hanoi(n, start, end):
    if n == 1:
        print(start, end)
    else:
        other = 6 - start - end
        hanoi(n-1, start, other)
        hanoi(1, start, end)
```

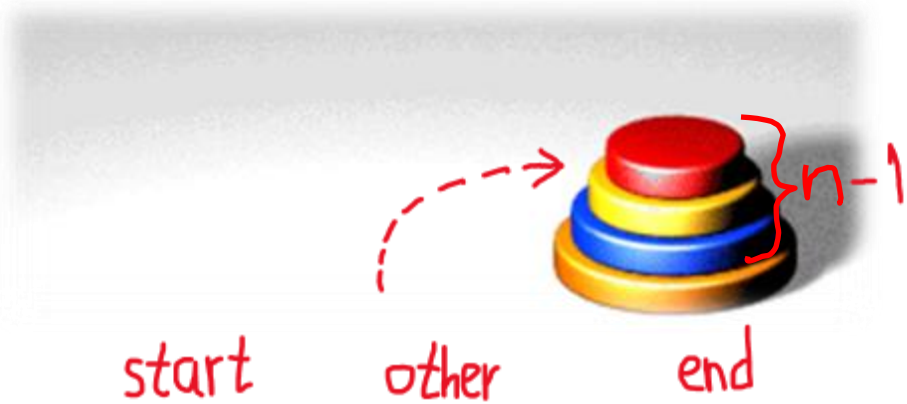



By André Karwath aka Aka - Own work, CC BY-SA 2.5,
https://commons.wikimedia.org/w/index.php?curid=85401

# Παράδειγμα: ο πύργος του Ανόι

- Πως θα μετακινήσουμε τον πύργο χωρίς να τοποθετήσουμε μεγαλύτερο δίσκο πάνω σε μικρότερο;

```
def hanoi(n, start, end):
    if n == 1:
        print(start, end)
    else:
        other = 6 - start - end
        hanoi(n-1, start, other)
        hanoi(1, start, end)
        hanoi(n-1, other, end)
```




By André Karwath aka Aka - Own work, CC BY-SA 2.5,
https://commons.wikimedia.org/w/index.php?curid=85401

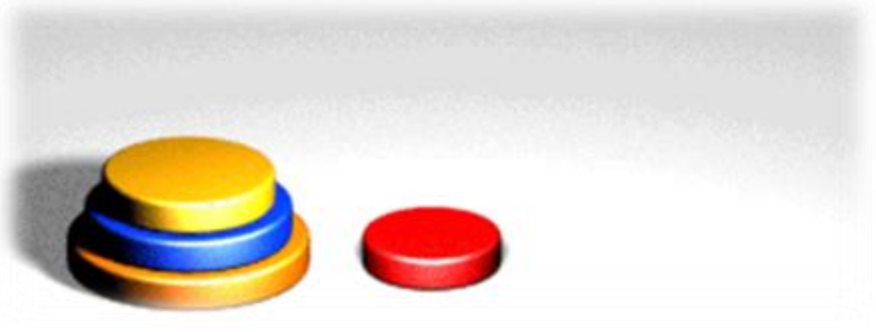
# Παράδειγμα: ο πύργος του Ανόι

```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```

1 2 3

# Παράδειγμα: ο πύργος του Ανόι

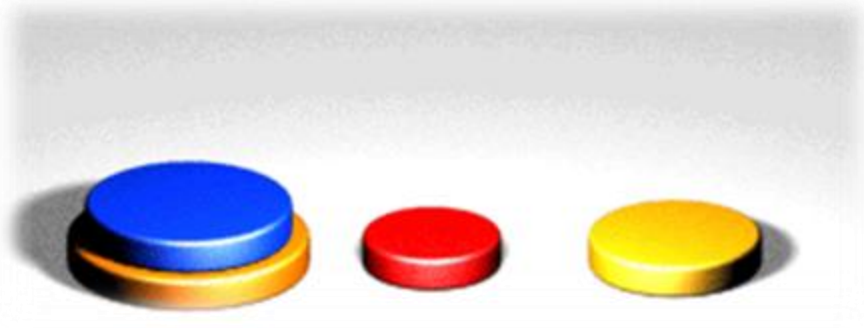
```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

# Παράδειγμα: ο πύργος του Ανόι

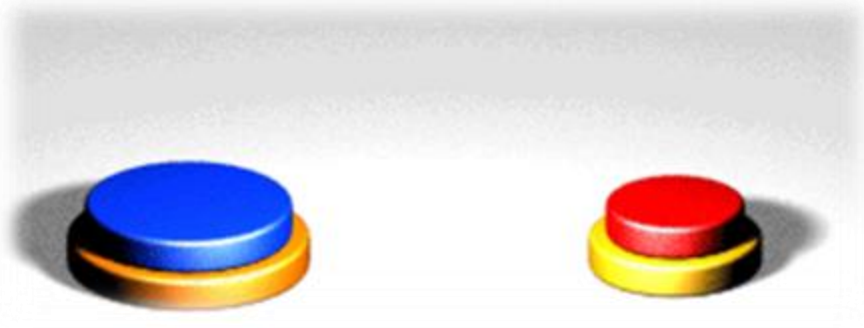
```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

# Παράδειγμα: ο πύργος του Ανόι

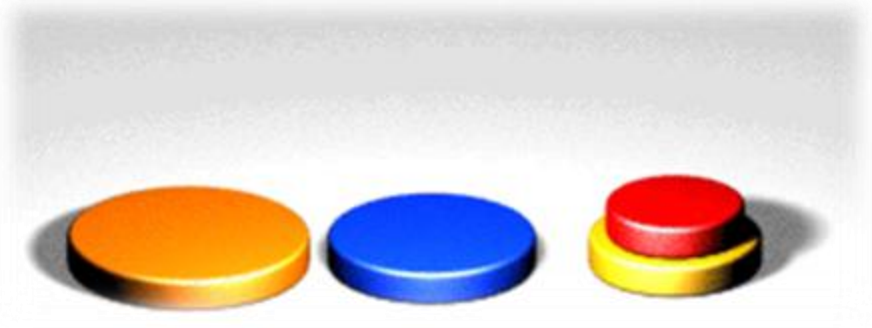
```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

# Παράδειγμα: ο πύργος του Ανόι

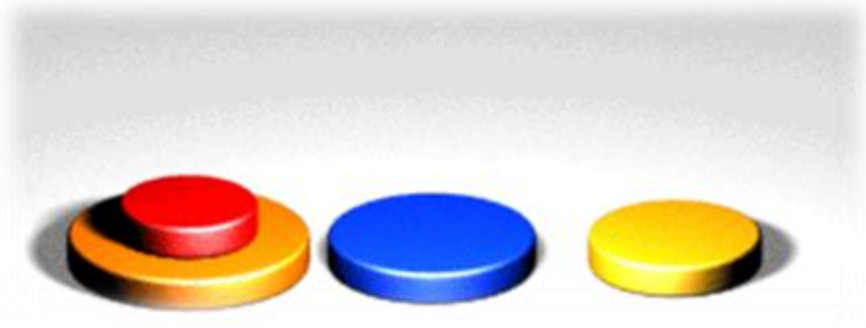
```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

# Παράδειγμα: ο πύργος του Ανόι

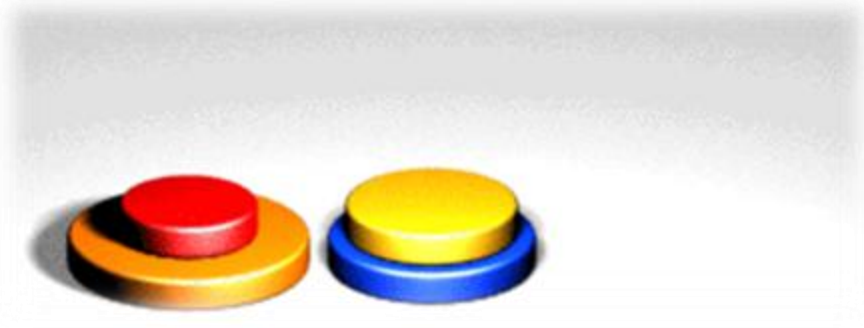
```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

# Παράδειγμα: ο πύργος του Ανόι

```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

# Παράδειγμα: ο πύργος του Ανόι

```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```

1 2 3

# Παράδειγμα: ο πύργος του Ανόι

```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```

1 2 3

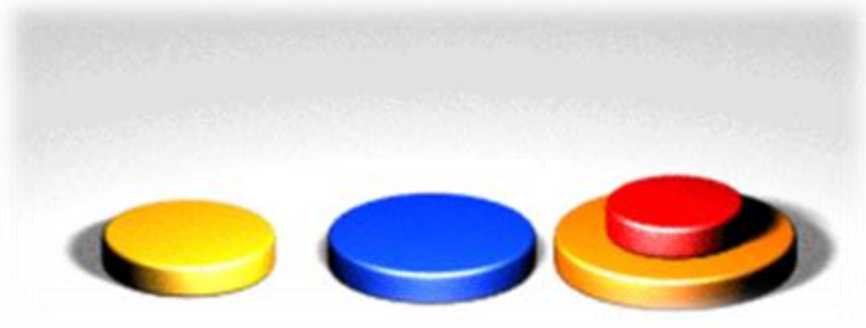
# Παράδειγμα: ο πύργος του Ανόι

```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```

1 2 3

# Παράδειγμα: ο πύργος του Ανόι

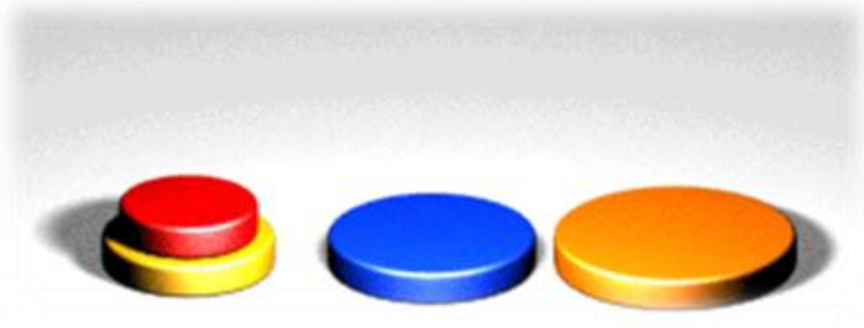
```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

# Παράδειγμα: ο πύργος του Ανόι

```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

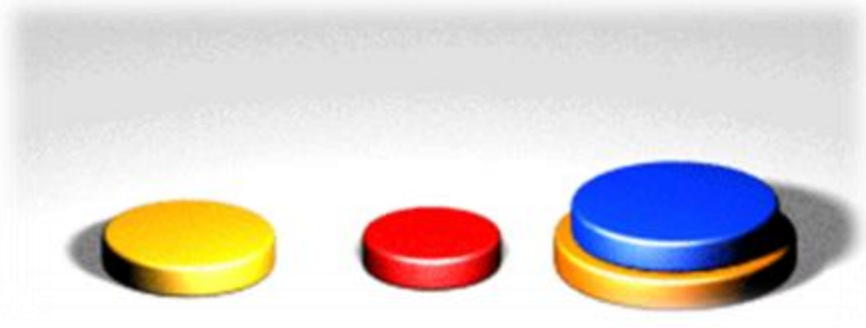
# Παράδειγμα: ο πύργος του Ανόι

```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```

1 2 3

# Παράδειγμα: ο πύργος του Ανόι

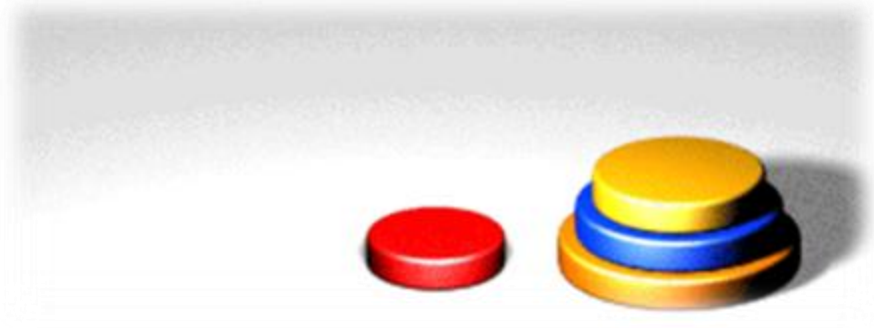
```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

# Παράδειγμα: ο πύργος του Ανόι

```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```



1 2 3

# Παράδειγμα: ο πύργος του Ανόι

```
>>> hanoi(4, 1, 3)
1 2
1 3
2 3
1 2
3 1
3 2
1 2
1 3
2 3
2 1
3 1
2 3
1 2
1 3
2 3
```

1 2 3

*Έκθεση χαρακτικής* (M. C. Escher, 1956)

- Επόμενη διάλεξη: Δεδομένα