Laboratorio 09

Programmazione - CdS Matematica

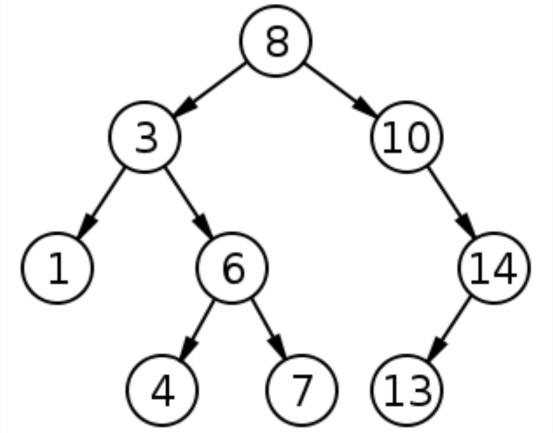
Ivano Lauriola

16 gennaio 2018

Binary Search Tree

Alberi binari di ricerca

Gli *alberi binari di ricerca* (binary search trees, **BST**), detti anche *alberi ordinati*, sono particolari alberi binari con la seguente proprietà:
il valore di ogni nodo è maggiore del valore di tutti i nodi del sotto-albero sinistro e minore del valore di tutti i nodi del sotto-albero destro.



Classe Albero

Definire la classe Albero (binario). È sufficiente definire il solo costruttore.

Classe Albero

Definire la classe Albero (binario). È sufficiente definire il solo costruttore.

```
class Albero:
    def __init__(self,val=None,sx=None,dx=None):
        self.val = val
        self.sx = sx
        self.dx = dx
```

Generazione BST

Definire la funzione `bst_ins(T,v)` (esterna alla classe Albero) che inserisca il valore v nell'albero T (anche vuoto) mantenendo la proprietà dei BST. La funzione deve ritornare l'albero con il nuovo valore inserito.

Generazione BST

Definire la funzione `bst_ins(T,v)` (esterna alla classe Albero) che inserisca il valore v nell'albero T (anche vuoto) mantenendo la proprietà dei BST. La funzione deve ritornare l'albero con il nuovo valore inserito.

```
def bst_ins(T,v):
    if not T:
        return Albero(v)
    if T.val>v:
        T.sx = bst_ins(T.sx,v)
    if T.val<v:
        T.dx = bst_ins(T.dx,v)
    return T
```

Funzione di stampa

- Definire una funzione `print_lista_albero(T)` che stampa a video gli elementi contenuti in un albero BST ordinati in modo crescente.

Funzione di stampa

- Definire una funzione `print_lista_albero(T)` che stampa a video gli elementi contenuti in un albero BST ordinati in modo crescente.

```
def print_lista_albero(T):
    if not T:
        return
    if T.sx:
        print_lista_albero(T.sx)
    print T.val
    if T.dx:
        print_lista_albero(T.dx)
```

Funzione di ricerca (valore esatto)

- Definire la funzione di ricerca `bst_search(T, v)` di un elemento v in un BST T: se l'elemento cercato esiste ritorna il sotto-albero che ha v come radice, altrimenti ritorna None.

Funzione di ricerca (valore esatto)

- Definire la funzione di ricerca `bst_search(T, v)` di un elemento v in un BST T: se l'elemento cercato esiste ritorna il sotto-albero che ha v come radice, altrimenti ritorna None.

```
def bst_search(T, v):
    if not T:
        return None
    if T.val==v:
        return T
    if T.val>v:
        return bst_search(T.sx,v)
    return bst_search(T.dx,v)
```

Calcolo del numero di ricorsioni

- Calcolare il numero di chiamate ricorsive necessario per la ricerca di un valore contenuto in un intervallo. Suggerimento: definire una variabile globale che faccia da contatore delle chiamate ricorsive alla funzione `bst_search_intervallo`.

```
conta_iterazioni = ...
def bst_search_intervallo(T, a, b):
    global conta_iterazioni
    ...
```

Calcolo del numero di ricorsioni

- Calcolare il numero di chiamate ricorsive necessario per la ricerca di un valore contenuto in un intervallo. Suggerimento: definire una variabile globale che faccia da contatore delle chiamate ricorsive alla funzione `bst_search_intervallo`.

```
conta_iterazioni = 0
def bst_search_intervallo(T, a, b):
    global conta_iterazioni
    conta_iterazioni = conta_iterazioni + 1
    if not T:
        return None
    if T.val>=a and T.val<=b:
        return T.val
    if T.val>b:
        return bst_search_intervallo(T.sx,a,b)
    return bst_search_intervallo(T.dx,a,b)
```

Esercizi su ricorsione ed alberi

Esercizio 1

Scrivere una funzione ricorsiva che, dato in input un intero positivo, ritorni True se e solo se tale valore è pari. **Non** si possono utilizzare operatori di moltiplicazione, divisione e resto.

Esercizio 1

Scrivere una funzione ricorsiva che, dato in input un intero positivo, ritorni True se e solo se tale valore è pari. **Non** si possono utilizzare operatori di moltiplicazione, divisione e resto.

```
def even(x):
    return True if not x else not even(x-1)
```

Esercizio 2

Descrivere `PRE` e `POST` condizioni della seguente funzione:

```
def c(a):
    return True if not a else c(a.sx) != c(a.dx)
```

Esercizio 2 - soluzione

`PRE`: `a` è un albero binario anche vuoto

`POST`: ritorna `True` sse il numero di nodi dell'albero è pari

`DIM`:

- `POST(c(None))` : 0 nodi nell'albero, ritorna `True`.
- `PRE(a.sx)` e `PRE(a.dx)` : valgono perché `a` non è vuoto, di conseguenza `a.sx` e `a.dx` sono due sotto-alberi validi.
- `POST(c(a))` : assumendo `a` non vuoto, il numero dei nodi in esso è pari se e solo se la somma del numero di nodi nei sottoalberi destro e sinistro è dispari, ovvero quando i risultati delle invocazioni ricorsive applicate ad `a.sx` ed `a.dx` sono diversi, cioè uno è `True` (pari) ed uno `False` (dispari). Siccome bisogna considerare anche il nodo rappresentato da `a`, interviene la negazione.

Esercizio 3

Dato un Albero binario creare una funzione `nodes_lev(tree, lev)` che restituisce il numero di nodi presenti al livello `lev` nell'albero.

Esercizio 3

Dato un Albero binario creare una funzione `nodes_lev(tree, lev)` che restituisce il numero di nodi presenti al livello `lev` nell'albero.

Soluzione:

```
class Tree():
    def __init__(self, val=None, sx=None, dx=None):
        self.val = val
        self.sx = sx
        self.dx = dx

def nodes_lev(tree, lev):
    if (tree == None or lev < 0): return 0
    if lev == 0: return 1
    return nodes_lev(tree.dx, lev-1)+nodes_lev(tree.sx,
        lev-1)
```

Esercizio 4

Dato un Albero binario creare una funzione ricorsiva `average(tree)` che restituisce la coppia (μ, σ) dove μ è il valore medio dei valori dei nodi presenti nell'albero `tree` e σ è il numero dei nodi contenuti in tale albero. Si assume che i valori all'interno dell'albero siano interi.

Esercizio 4 - Soluzione

```
class Tree():
    def __init__(self, val=None, sx=None, dx=None):
        self.val = val
        self.sx = sx
        self.dx = dx

def average(tree):
    if tree == None: return (0.0, 0)

    sm, sc = average(tree.sx)
    dm, dc = average(tree.dx)

    cnt = sc + dc + 1
    return (sm*sc + dm*dc + tree.val) / float(cnt), cnt
```

Esercizi di ripasso

Esercizio 5

Scrivere un descrittore di lista che crea una matrice (lista di liste) quadrata con 7 righe, tale che ogni elemento rappresenta la sua distanza dalla diagonale.
Esempio, matrice 3x3: [[0,1,2],[1,0,1],[2,1,0]].

Esercizio 5

Scrivere un descrittore di lista che crea una matrice (lista di liste) quadrata con 7 righe, tale che ogni elemento rappresenta la sua distanza dalla diagonale.
Esempio, matrice 3x3: [[0,1,2],[1,0,1],[2,1,0]].

Soluzione:

```
[[abs(i-j) for i in range(7)] for j in range(7)]
```

Esercizio 6

Date le seguenti istruzioni, dire senza eseguire, quale sarà il valore contenuto nella variabile `t`.

```
x = 1
y = 2
t = (x, y*x, x*x, [y,3])
x = x * t[2]
t[3][1] = x
t[3].extend([x, y])
y = x**2 + t[1]
t[3][3] -= 1
```

Esercizio 6

Date le seguenti istruzioni, dire senza eseguire, quale sarà il valore contenuto nella variabile `t`.

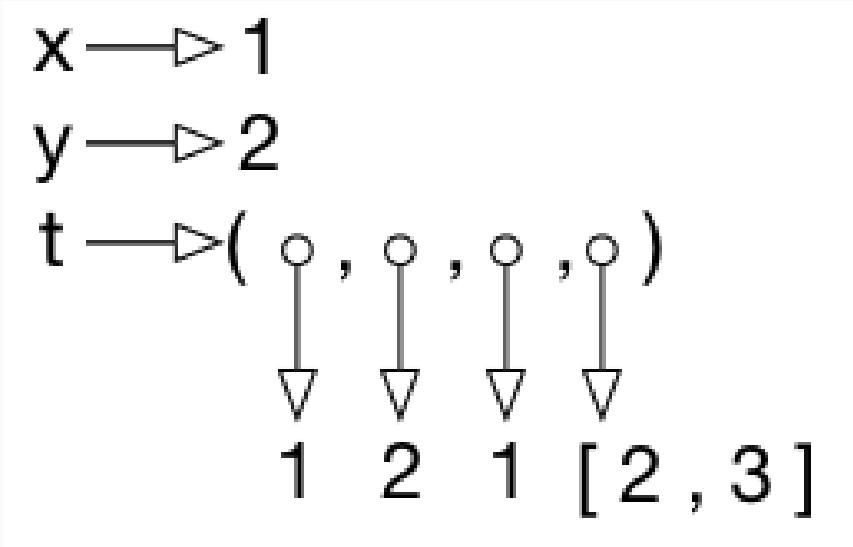
```
x = 1
y = 2
t = (x, y*x, x*x, [y,3])
x = x * t[2]
t[3][1] = x
t[3].extend([x, y])
y = x**2 + t[1]
t[3][3] -= 1
```

Soluzione:

```
t = (1, 2, 1, [2, 1, 1, 1])
```

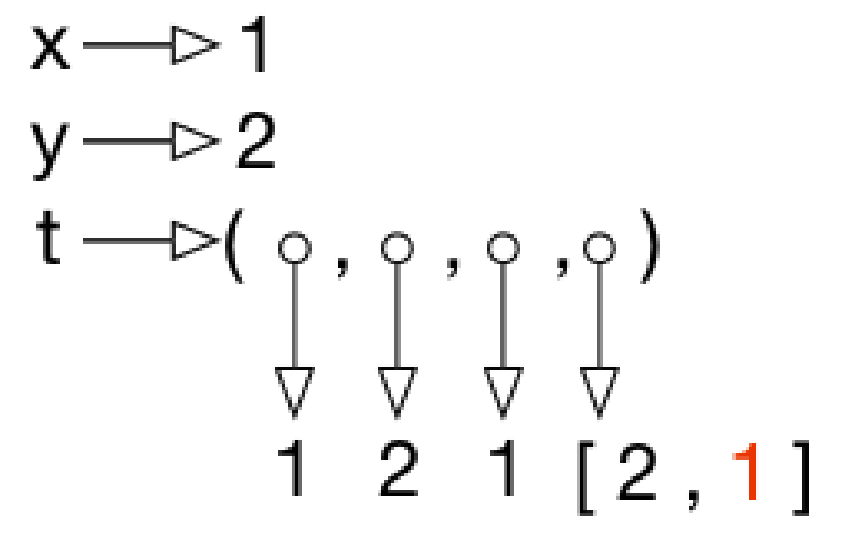
Esercizio 6 - Spiegazione (1/5)

```
x = 1
y = 2
t = (x, y*x, x*x, [y,3])
```



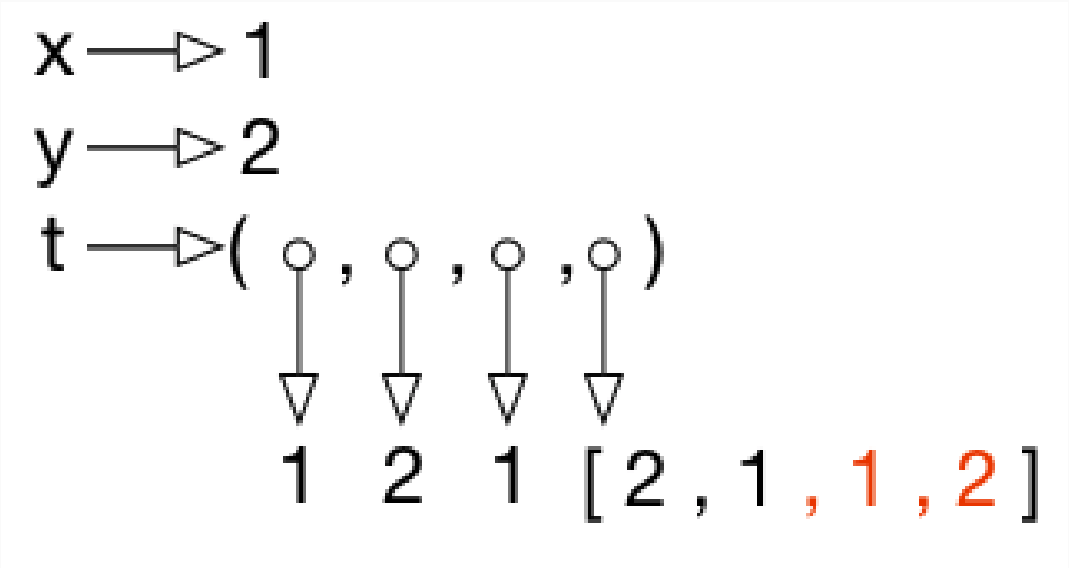
Esercizio 6 - Spiegazione (2/5)

```
x = x * t[2]
t[3][1] = x
```



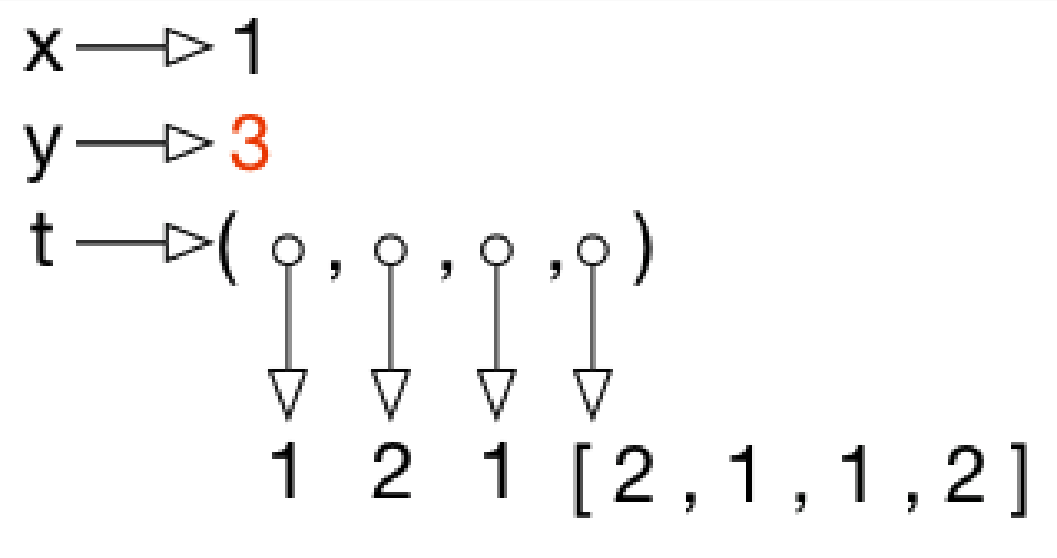
Esercizio 6 - Spiegazione (3/5)

```
t[3].extend([x, y])
```



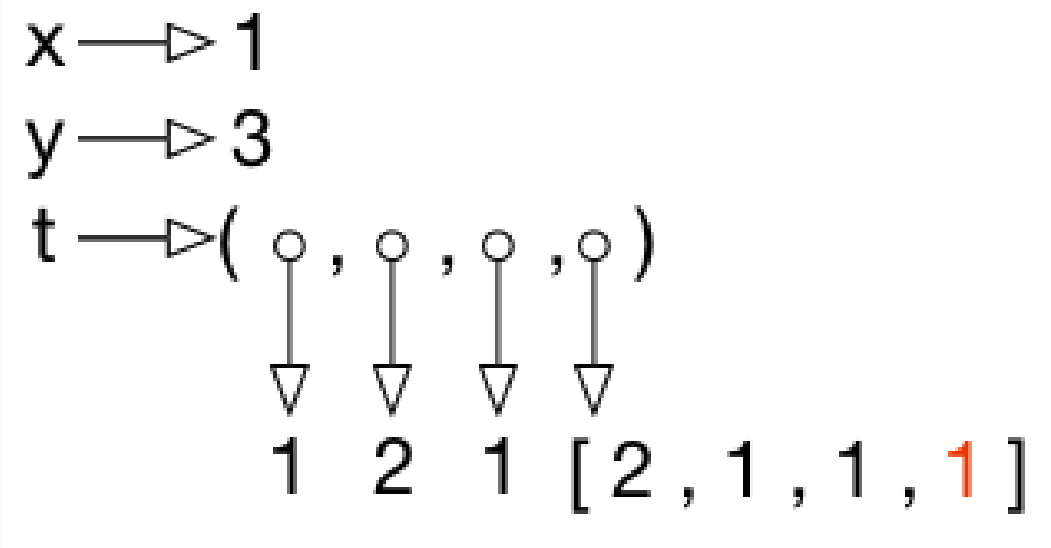
Esercizio 6 - Spiegazione (4/5)

```
y = x**2 + t[1]
```



Esercizio 6 - Spiegazione (5/5)

```
t[3][3] -= 1
```



Esercizio 7

Date le seguenti istruzioni, dire senza eseguire, quale sarà il valore contenuto nella variabile `x`.

```
x = [42]
for i in range(4):
    y = x
    x[0] = i
    x = [x, y]
```

Esercizio 7

Date le seguenti istruzioni, dire senza eseguire, quale sarà il valore contenuto nella variabile `x`.

```
x = [42]
for i in range(4):
    y = x
    x[0] = i
    x = [x, y]
```

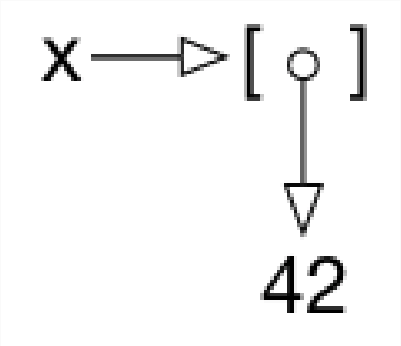
Soluzione:

```
x = [[3, [2, [1, [0]]]], [3, [2, [1, [0]]]]]
```

Esercizio 7 - Spiegazione (1/5)

Situazione iniziale:

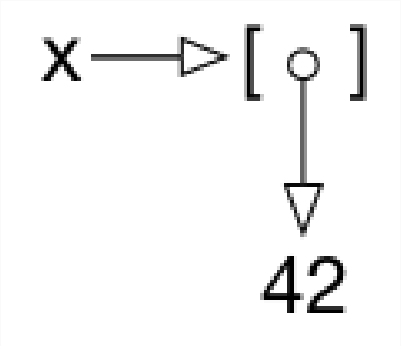
```
x = [42]
```



Esercizio 7 - Spiegazione (1/5)

Situazione iniziale:

```
x = [42]
```



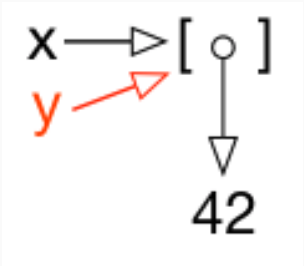
Ciclo **for** per $i \in \{0, 1, 2, 3\}$:

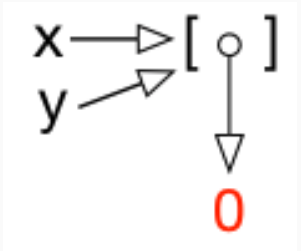
```
for i in range(4):
```

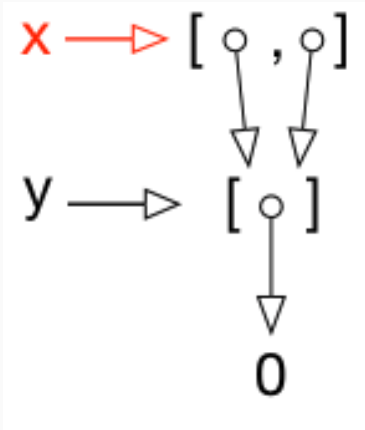
Esercizio 7 - Spiegazione (2/5)

All'interno del ciclo for con $i = 0$

```
y = x                    x[0] = i                    x = [x, y]
```



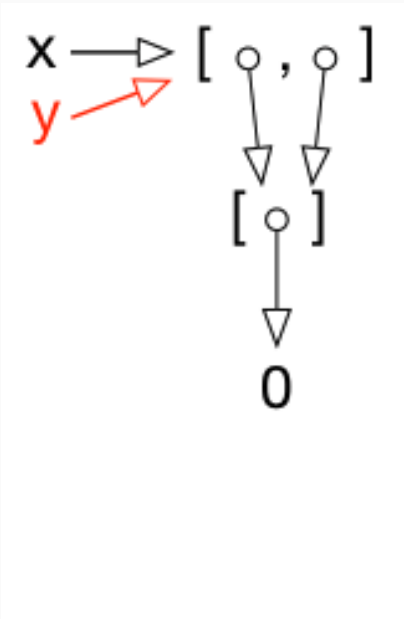


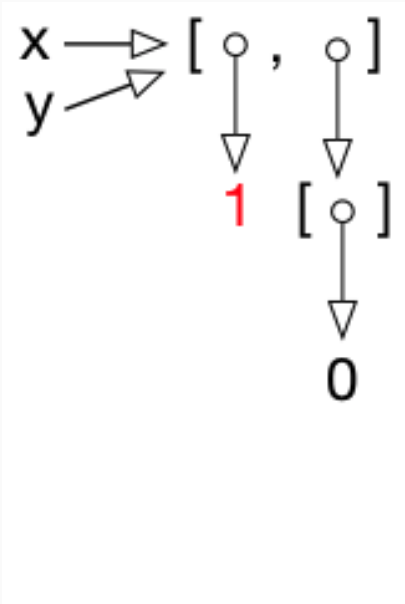


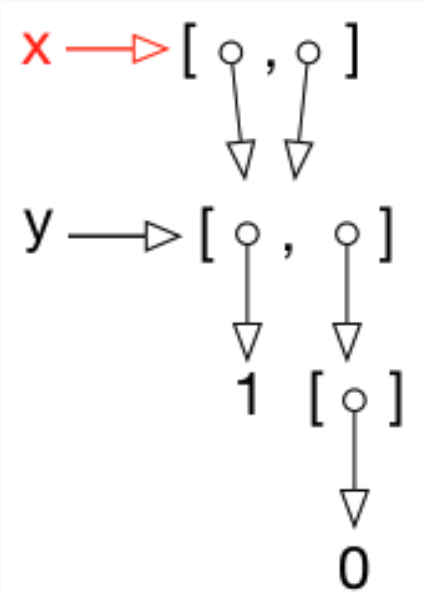
Esercizio 7 - Spiegazione (3/5)

All'interno del ciclo for con $i = 1$

```
y = x                    x[0] = i                    x = [x, y]
```



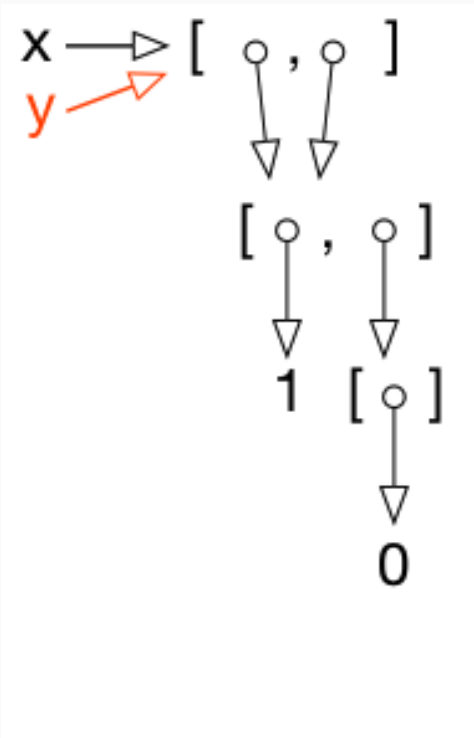


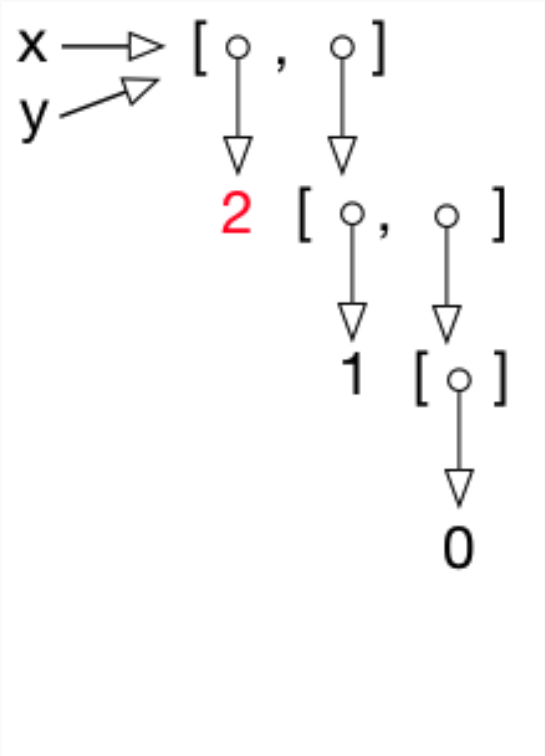


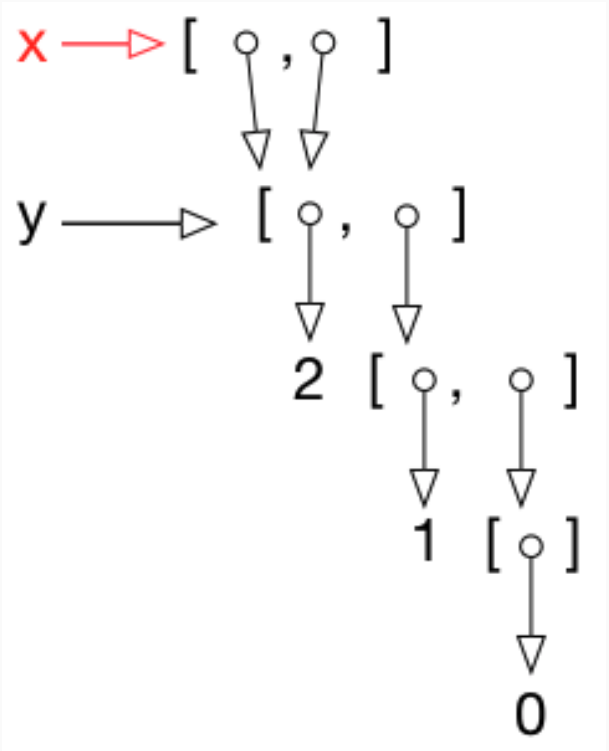
Esercizio 7 - Spiegazione (4/5)

All'interno del ciclo for con $i = 2$

```
y = x                    x[0] = i                    x = [x, y]
```



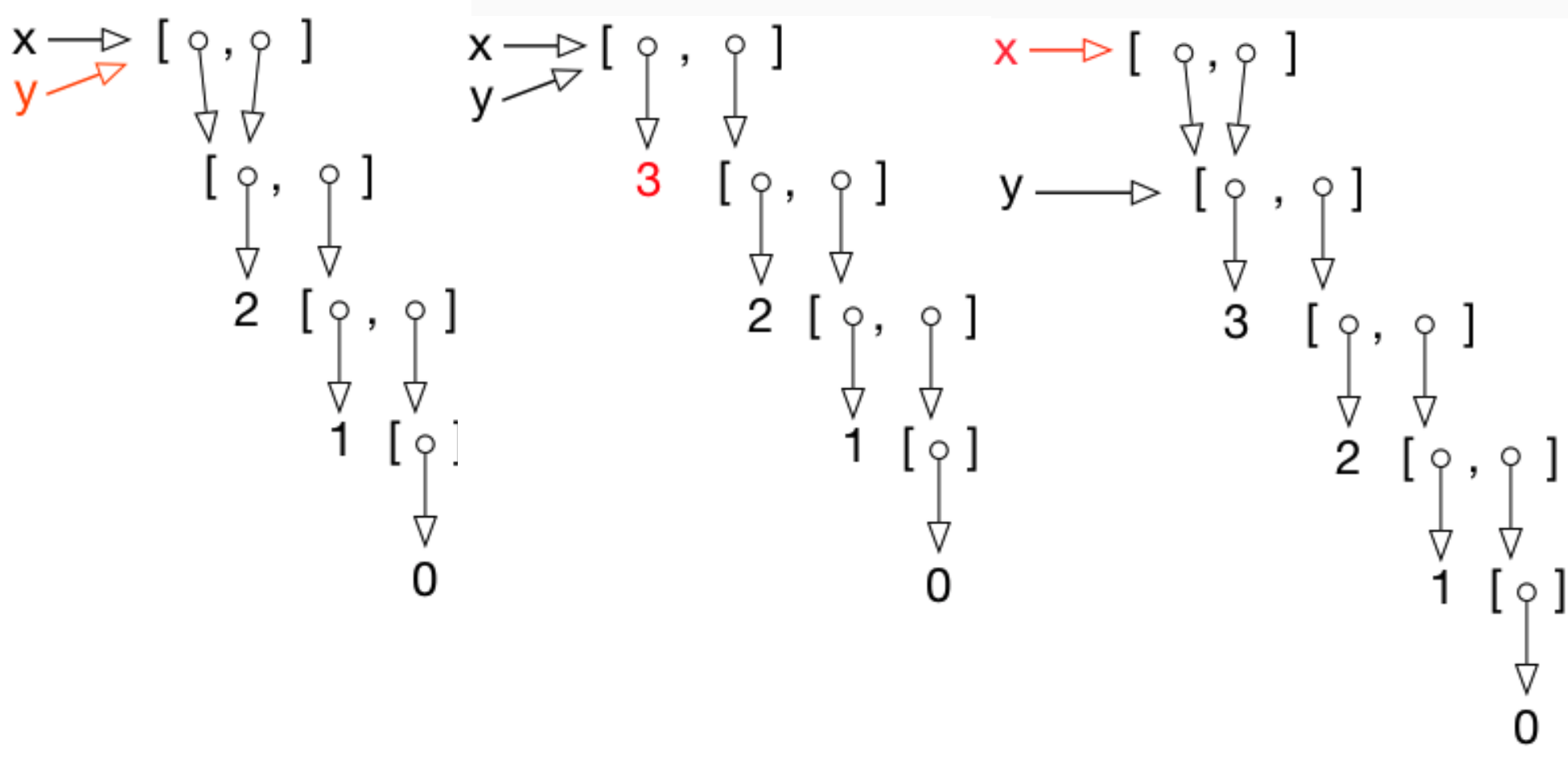




Esercizio 7 - Spiegazione (5/5)

All'interno del ciclo for con $i = 3$

```
y = x                    x[0] = i                    x = [x, y]
```



Esercizio 8

Creare una classe che gestisca una partita a Sette e Mezzo. Regole base:

- si gioca con un mazzo di carte "all'italiana", ovvero 40 carte con 4 semi: bastoni, spade, denari e coppe;
- ogni carta vale quanto il numero che rappresenta ad eccezione delle figure (i.e., fante(8), cavallo(9) e re(10)) che valgono $\frac{1}{2}$;
- il gioco si svolge tra giocatore e banco: vince chi possiede come somma delle proprie carte il valore più vicino ($\leq$) al $7\frac{1}{2}$. Chi lo supera "sballa" e perde;
- il giocatore riceve carte fintanto che lo vuole o sballa; il banco (CPU) accetta carte fintanto che non ha un valore $\geq 5\frac{1}{2}$;

Esercizio 8

Viene richiesto di:

- creare una classe che gestisca il gioco per un numero di turni fissato;
- ogni turno deve:
  - assegnare una carta a banco e giocatore (il giocatore deve conoscere anche la carta del banco);
  - chiedere all'utente se vuole carta o si ferma, fino a che non si ferma o sballa;
  - successivamente, se il giocatore non ha sballato, pesca le carte al banco fino a che non ha un valore $\geq 5\frac{1}{2}$.
- in caso di stessi punti (senza essere sballati) vince il banco. Se il giocatore sballa, il banco vince senza pescare altre carte;
- finiti i turni: dire chi ha vinto più mani.

Esercizio 8

La soluzione è liberamente scaricabile al link:

`http://www.math.unipd.it/~mpolato/didattica/7emezzo.py`