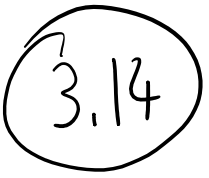
OGGETTI

giovedì 25 ottobre 2018 09:48

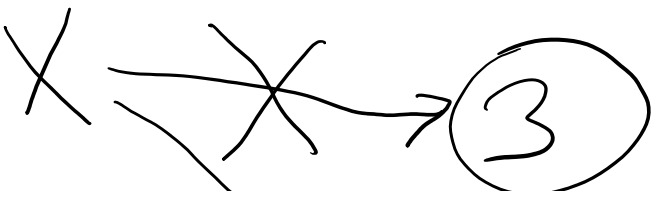
- identità
- tipo
- valore

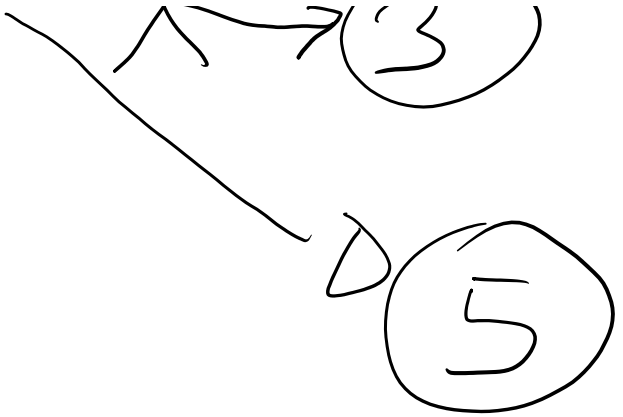
id (3)

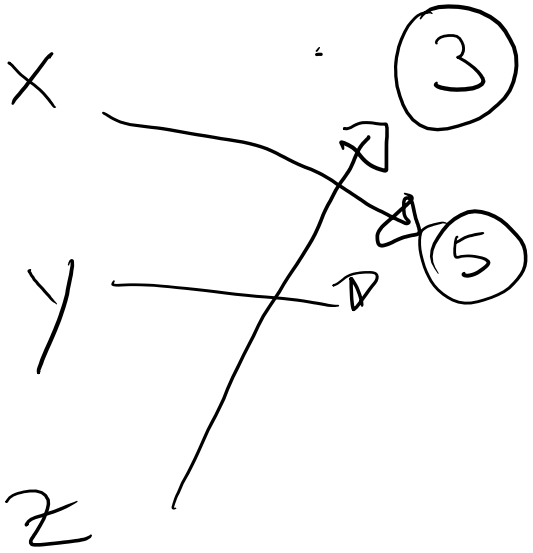
int
float
str



RIFERIMENTI







X = 3

Y = 5

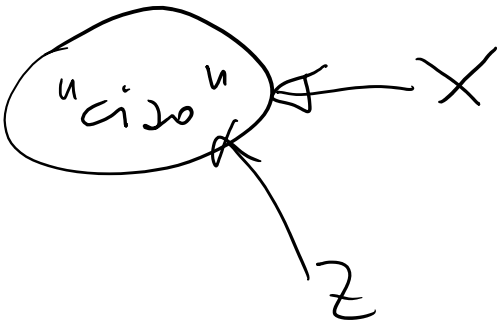
Z = X

X = Y

X = "ciao"

Z = X

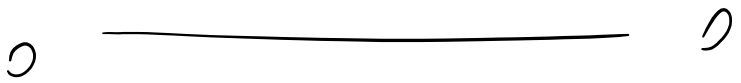
Z is X



2 Primo, contiene alfabetico

2 Solo alfanumerici + _ a31_bis

3 Escluse keywords di Python

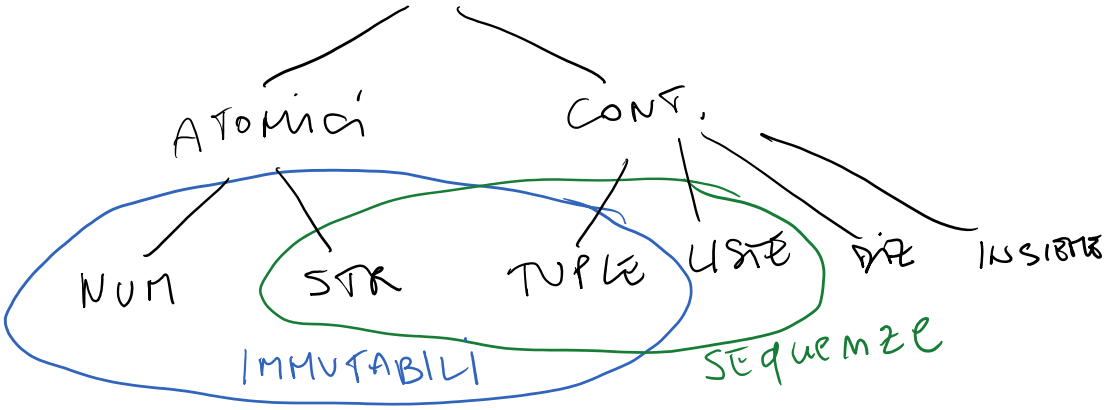


TIPI PRIMITIVI

NUMERICI (int, float, bool, complex)
STRINGHE (str)
Tupla (tuple)
LISTA (list)
DIZIONARI (dict)
INSIEME (set)

- MOD. DI MEMORIZZAZIONE
  - ATOMICO < STRINGHE, NUMERICI
  - CONTENITORE < TUPLE, LISTE, DIZ, INS.
- MOD. DI AGGIORNAMENTO
  - MUTABILE (LISTE, DIZ, INSIEMI)
  - NON MUTABILE (NUM, STR, TUPLE)
- MOD. ACCESSO
  - ITERABILE (STR + CONT)
  - NON ITERABILI (NUM)
  - SEQUENZIALI [5, -1, 14, 28]
  - ASSOCIATIVI

TIPI



TIPI ATOMICI

NUMERICI (INT

STRINGHE

bool ⊂ int ⊂ float ⊂ complex

4 + (2+3j)

6+3j float(3+2j)

STRINGHE

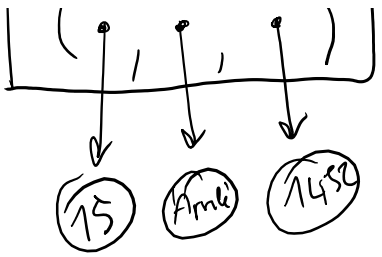
"abcdef".capitalize() → "Abcdef"

upper()

lower()

(15, 'April', 1452)

data → (,,,)

date → 

Operatori di formato

stringa di formato % tupla

in caso mi aspetto %s in % ("Gianni",)

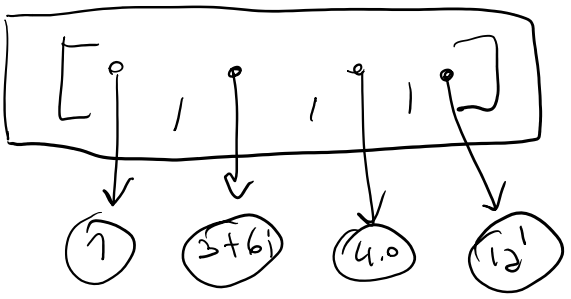
in base se % d e % d e' % d) %

(3, 4, 7)

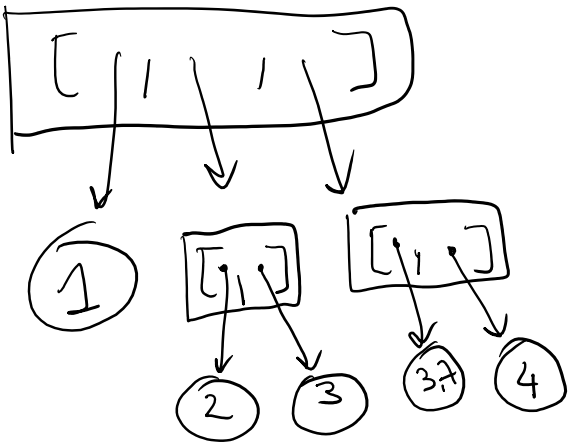
LISTE

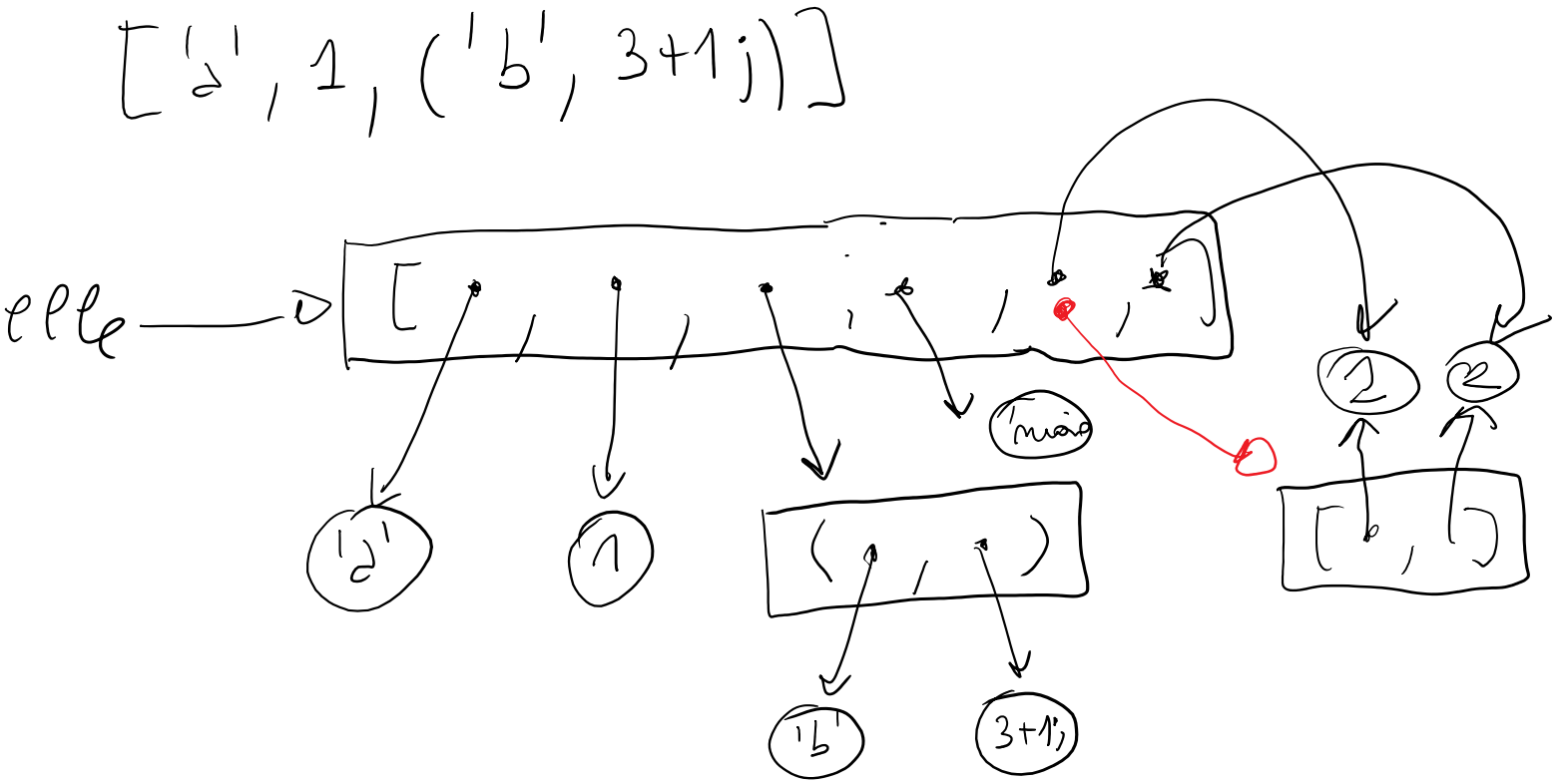
→ [1, 3+6j, 4.0, 'a']

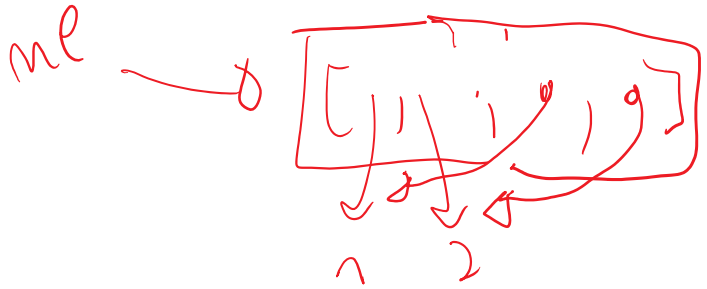
[]



[1, [2, 3], [3.7, 4]]

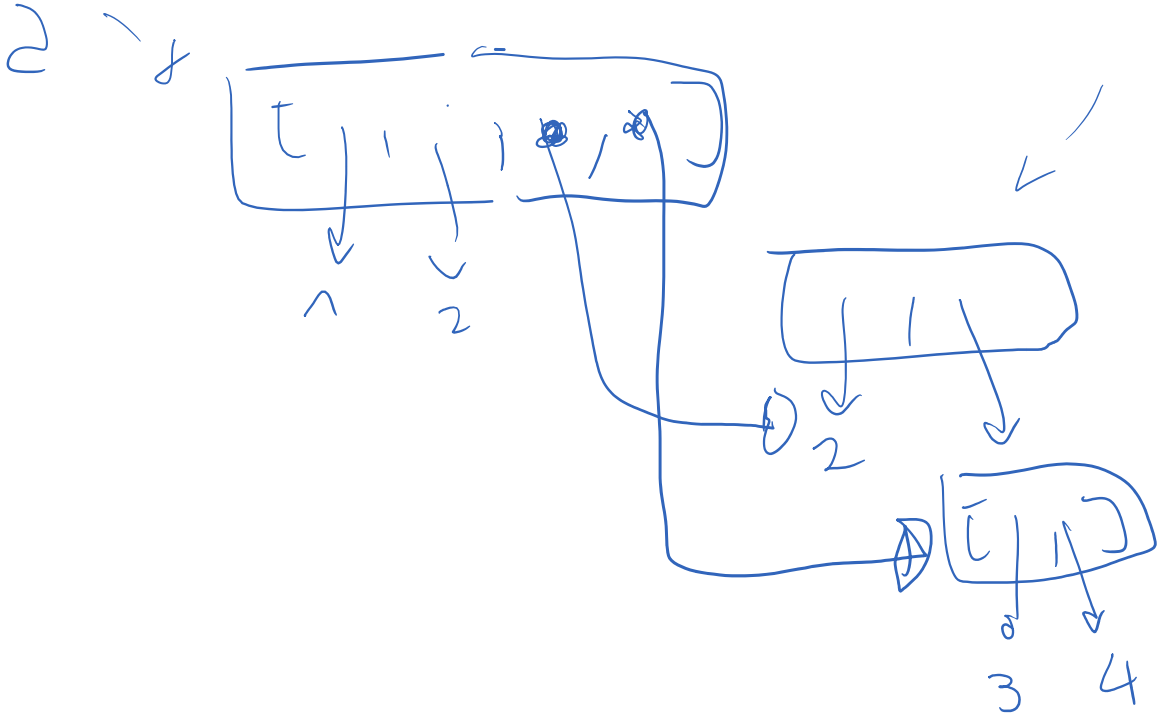






elle.reverse

```
a = [1, 2]
a.extend([2, [3, 4]])
```



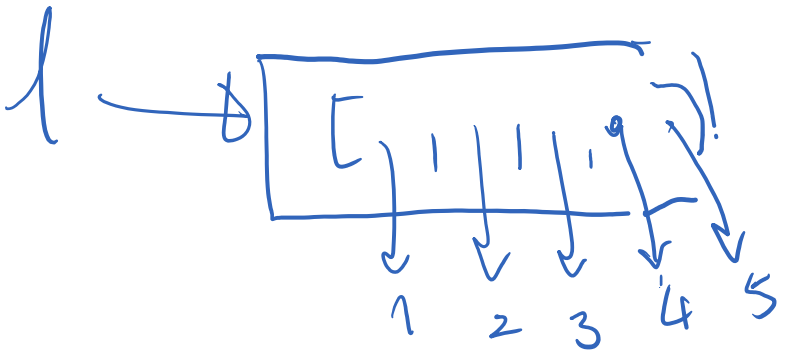
Finiamo le liste e parliamo di sequenze..

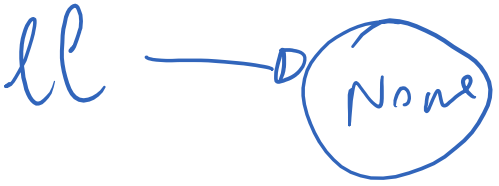
mercoledì 31 ottobre 2018 11:09

Split

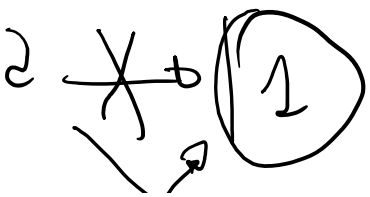
S.split (<sep>)

Default: ' ' '\n'

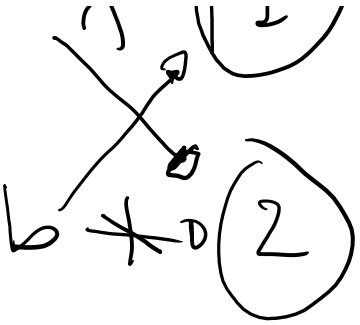




<sep>.join (lista)



a,b = 1,2



1 $a = b$

$b = a$

2 $a, b = b, a$

OPERAZIONI X SEQUENZE

mercoledì 7 novembre 2018 10:59

```
a = 3
b = 2
a,b = b,a
```

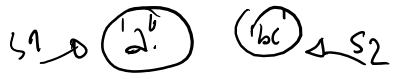
```
a = 3
b = 2
c = b
```

(a → 3, c → 3, b → 2)

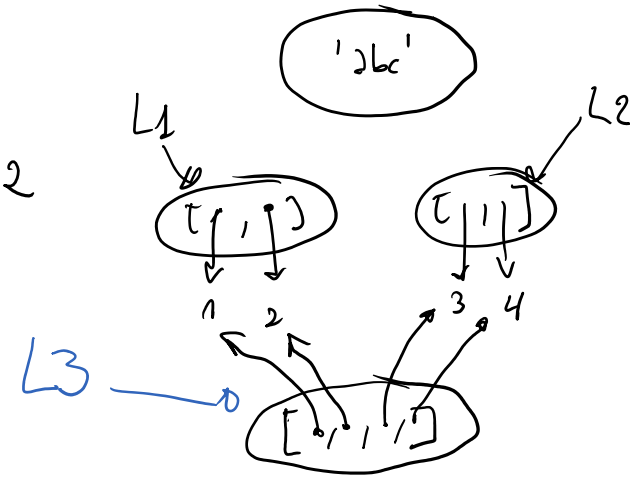
```
a = 3
b = 2
temp = a
a = b
b = temp
```

CONCATENAZIONE

S1 + S2



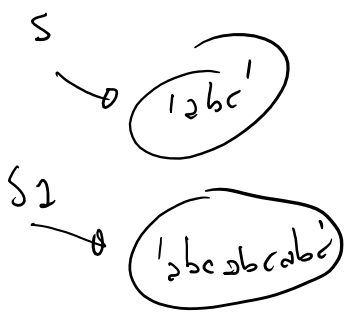
L3 = L1 + L2



RIPETIZIONE

S = 'abc'

S1 = S*3



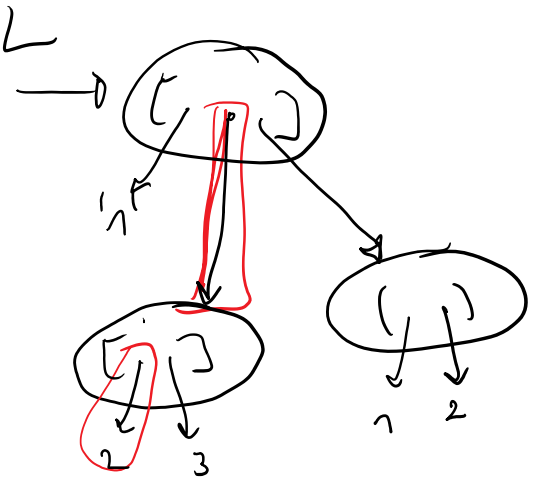
INDICIZZAZIONE

s[0], s[1], ...

L = ['1', [2,3], (1,2)]

L[1] → [2,3]

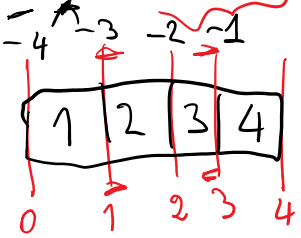
L[1][0]



SLICING

seq[start : end]

[1,2,3,4][1:3] → [2,3]



seq[start :]

seq[:] = seq[0, len(seq)]

s = '1234'

s[-1] → '4'

s[2:-1] → '3'

STRIDING

seq [start : end : step]

L = [1, 2, 3, 4, 5]

L[::2]

L[-1:-3:-2] → [5]

start

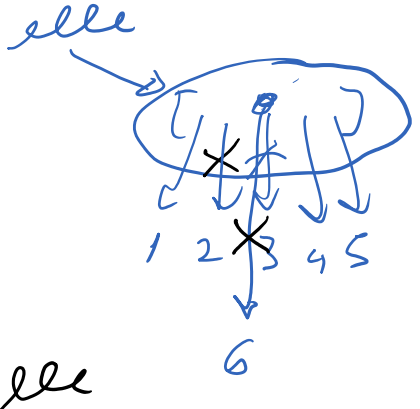
elle = [1, 2, 3, 4, 5]

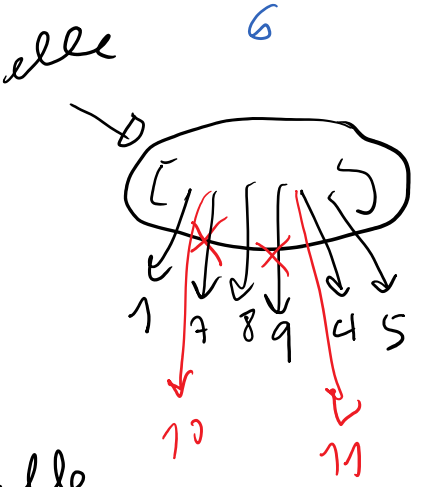
elle[2] = 6

elle[1:3] = [7, 8, 9]

elle[1:5:2] = [10, 11]

elle[-1:-4:-2] = ('a', 'b')





elle[1:-2] = 'cde'

a = 'cde'

elle[5] = 'cde'

elle[:1] = ['cde']

elle

del elle[3]

del elle[:2]

del eeee[::2]

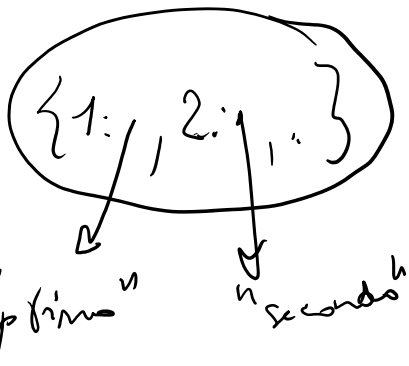
DIZIONARI

vuoto = {} vuoto = dict()

pieno = {1: "primo", 2: "secondo", 5: "quinto"}

pieno[2] → 'secondo'

pieno[4] = 'quarto'



del pieno[4]

d = {'k1': 1, 'k2': 2}

NO EC

d.keys() → ['k1', 'k2']

d.values() → [1, 2]

d.items() → [('k1', 1), ('k2', 2)]

EC d.clear()

EC d.clear() ⟶ {}

INSIEMI

vuoto = set()

pieno = set(["uno", "mar", "mer"])

bye = set("bye") set(["bye"])

set(["bye"]) ~~set(["bye"])~~

= set("byebye guy")

disp = set([1,3,5,7,9])

pari = set([2,4,6,8])

disp | pari ⟶ set([1,...,9])

mult3 = set([3,6,9]) {3,9}

disp & mult3 ⟶ set([3,9])

pari - mult3 —> set ([2,4,8])

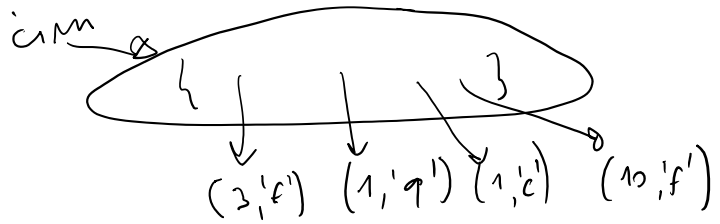
disp^ mult3 —> set ([1,5,7,6])

s.add ('2')

s.remove (3)

OPERAZIONI su ITERABILI

venerdì 9 novembre 2018 10:30



DESCRITTORI di LISTA

⇒ [f(x) for x in IT]

range (start, end)

range(2,7)

↓

[2,3,4,5,6]

⇒ [x*x for x in range(1,11)]

[f(x) for x in IT if g(x)]

[x for x in range(1,101) if x%2==0]

range(1,101) → IT

x%2==0 → g(x)

mat = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{bmatrix}$$

[[r[j] for r in mat] for j in range(4)]

[[...]

f(j)

[f(x1, x2) for x1 in IT1 if g(x1)
for x2 in IT2(x1) if h(x1, x2)]

[item for riga in mat for item in riga]

ES. 3.8

N : int
L : lista di interi

$2 \leq n \leq N$

N = 3 , L = [4, 6, 9, 10, 11, 12]

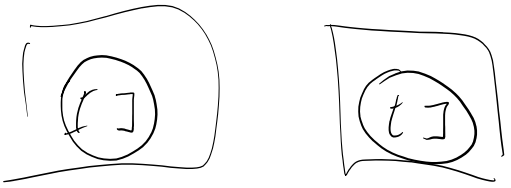
ris = [(4, 32), (3, 27)]

ESP. BASICHE bool(obj) → True / False

- None (NoneType)
- False
- 0, 0L, 0.0, 0.0j
- '', (), [], {}, set([])

IDENTITÀ

x IS y



COMPARAZIONE

3.78 < 4.11

APPARTENENZA

obj in it

CONNETTIVI LOGICI

A and B SHORT CIRCUIT

A or B

ESP. CONDIZIONALE

(EXPT if COND else EXPF)

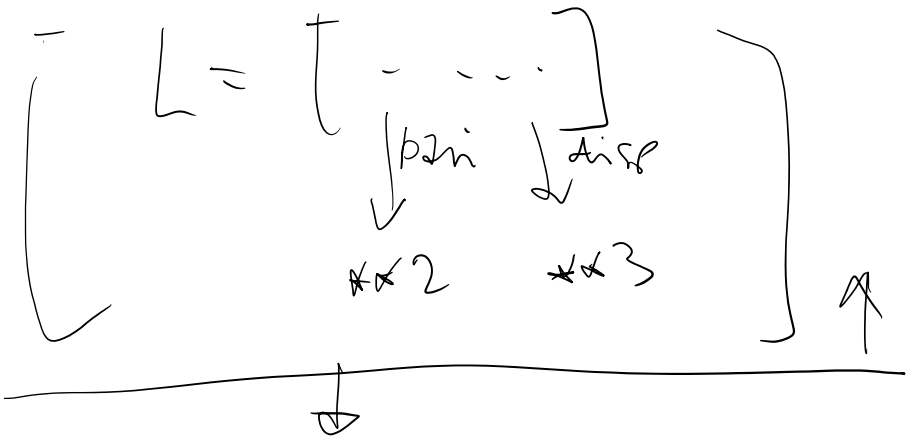
[exp(i) for i in IT if i>0]

[(x if x>0 else -x) for x in [-3,8,2,4,6,1]]

[+3,8,2,4,6,1]

[x for x in (c x>0)]

[x for x in (if x > 0)]

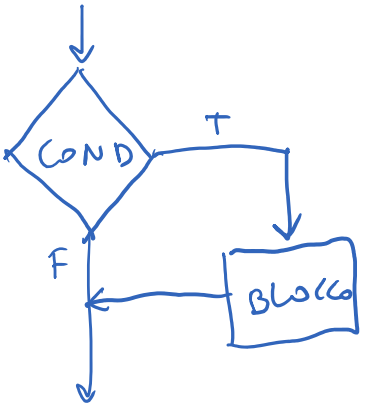


solo pari

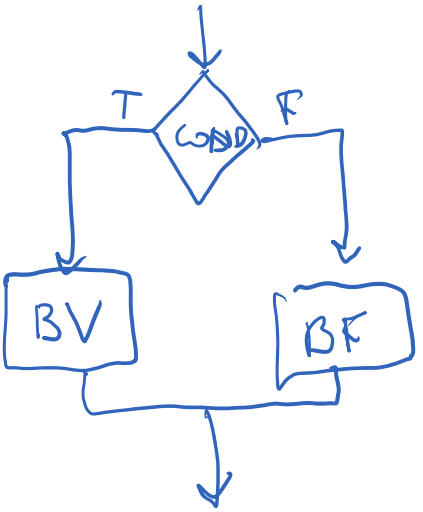
[y for y in [x**2 if x%2 == 0 else x**3 for x in L]
if y%2 == 0]

raw_input (prompt) → str

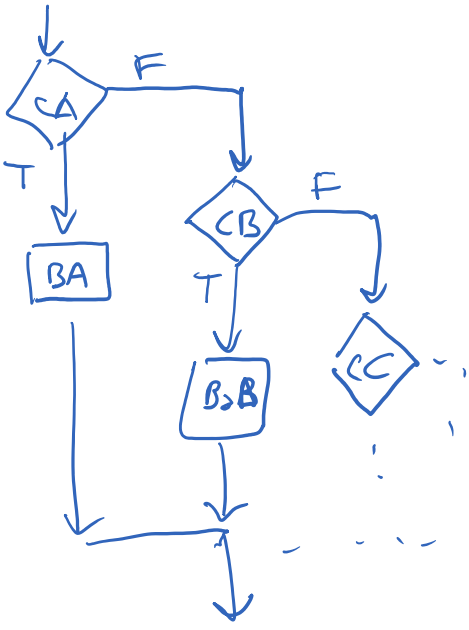
(i) if COND :
    BLOCCO



(ii) if COND :
    BLOCCO_SE_VERA
else :
    BLOCCO_SE_FALSA



(iii) if COND_A :
    BLOCCO_A_VERA
elif COND_B :
    BLOCCO_A_FALSA_B_VERA
...
else :
    BLOCCO_TUTTE_FALSE



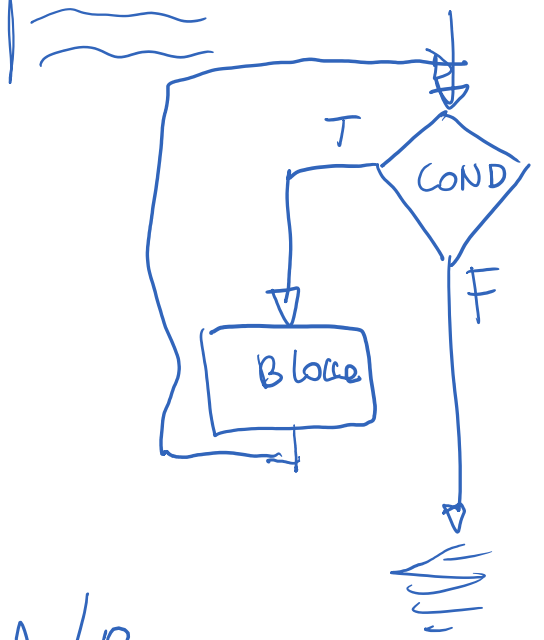
- 3 + 4
- 7 * 2
- 10.2 / 2
- 10 - 4

- X Lettura dell'espressione
- X operandi e operazione
- X Esegue op sugli operandi
- X stampare ris.

COSTRUTTI DI ITERAZIONE

giovedì 22 novembre 2018 10:04

```
while COND:
    BLOCCO
```




```
a = 2/0
while a>0:
    b=3
    b+=1
```

```
i = 0
while i < 100:
    Blocco
    i = i+1
```

Inserire 3
Inserire 4
Inserire -1

```
Inserire 5

num = int(raw_input("Inserire: "))
while num != 5:
    num = int(raw_input("Inserire: "))
```

```
s = "bcdef"
solo_consonanti = True
i = 0
while solo_consonanti and i < len(s):
    if s[i] in "aeiou":
        solo_consonanti = False
    else:
        i += 1
```

```
una_vocale = False
i = 0
while i < len(s) and not una_vocale:
```

```
if S[i] in "aeiou":
    une_vocale = True
else:
    i += 1
```

ITERATORE FOR

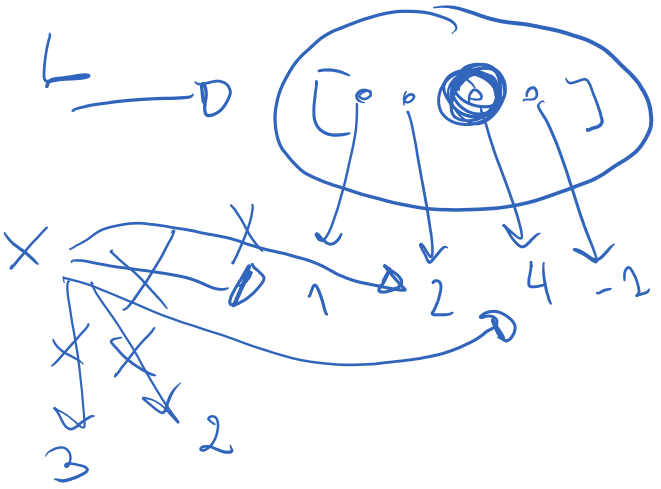
```
for x in IT:
    BLOCCO
```

```
for x in range(4):
    print("Hello!")
```

```
somma = 0
for x in range(1, 101):
    Somme += x
```

```
L = [1, 2, 4, -1]
for x in L :
    print x
    x = x + 1
```

L ?




```
for x in reversed(IT):

for x in sorted(IT):

for x in range(len(L)):
```

```
L[x] += 1
```

L ?

```
for i,c in enumerate("ciao"):
    print i, c
```

```
0 c
1 i
2 a
3 o
```

```
for i,c,d in LISTA:
```

```
d = {}
for i,c in enumerate(L):
    d[i] = c
```

```
WHILE COND :
    BLOCCO
```

```
FOR X IN IT :
    BLOCCO
```

Es 4.7

- RICHIEDI UN NUMERO INTERO POSITIVO n
- STAMPARE I DIVISORI DI n

```
n = int(raw_input("Inserire numero: "))
while n <= 0 :
    n = int(raw_input("Inserire numero: "))
```

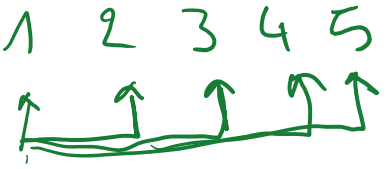
```
for i in range(2, n+1):
    if n % i == 0 :
        print i
```

n % i == 0 → mod n%i

- SOMMA dei primi n numeri interi

n da inserire

```
i, somme = 1, 0
while i <= n :
    somme += i
    i += 1
```




```
for i in range(1, n+1):
    somme += i
```

```
L = range(1, n+1)
somme = sum(L)
```

4.3

numero: 1
numero: -4
numero: 2
numero: 5

numero: 0

Somma: 4
Media: 1.0
Minimo: -4
Massimo: 5

```
x = int(raw_input("Numero: "))
Minimo = Massimo = x
n = 1
somma = x
while x != 0:
    x = int(raw_input("Numero: "))
    if(x != 0):
        n += 1
        somma += x
        if x < Minimo:
            Minimo = x
        if x > Massimo:
            Massimo = x
```

```
print "Somma: %d" % somma
print "Media: %f" % float(somma)/n
print "Minimo: %d" % minimo
print "Massimo: %d" % massimo
```

N = 20

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~
~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~

HINT #1

P = [True/False]
0 n

(True if i > 1 else False)

n = 100

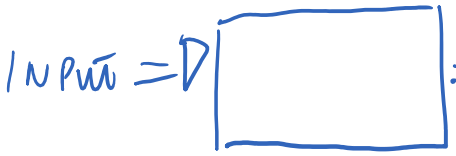
P = [True for i in range(n+1)]

P[0] = P[1] = False

FUNZIONI

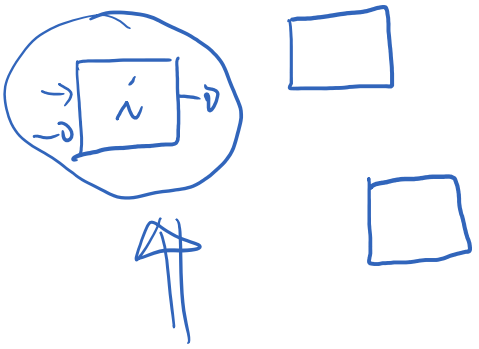
giovedì 29 novembre 2018 10:40

lem (IT)



$y = f(x)$

MODULARI



MODIFICABILE

RIUSABILE

LEGGIBILE

COMPATTI

- PRE-CONDIZIONI
- POST-CONDIZIONI

def nome-funzione (<par-formali>):

... ...

stringa di doc.

CORPO DELLA FUNZIONE

```
def saluti():
    print 'Salve!'
```

```
>> saluti()
Salve!
```

un_nome

```
def saluti_a(un_nome):
    print 'Salve %s!' % un_nome
```

```
nome_funzione(<par. attuali>)
```

```
saluti_a("Gino")
saluti_a("Luigi")
saluti_a("Gino" + "Luigi")
```

un_nome → "GinoLuigi"

return EXP

```
def f(n):
    corpo
    return EXP
```

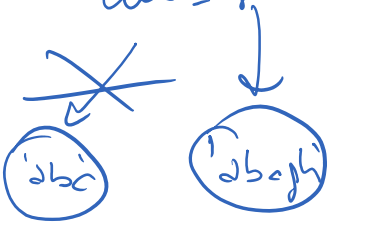
```
>> f(...)
>> a = f()
>> print f(1) + f(n)
```

```
def valore_assoluto(x):
    if x<0:
        return -x
    return x

print valore_assoluto(-9)
```

```
def is_primo(n):
```

```
for i in range(2, n/2):        2 ≤ i ≤ n/2
    if n % i == 0:
        return False
return True
```

```
una_globale = 'abc'
def fun2():
    una_locale = 'def'
    print una_globale + una_locale

fun2()          -> stampa 'abcdef'
print una_globale  -> 'abc'
una_globale += 'gh'
fun2()          -> stampa 'abcghdef'
```




```
def f(a)
    def g(b):
        BODY_G

    BODY_F
    g(##)
```

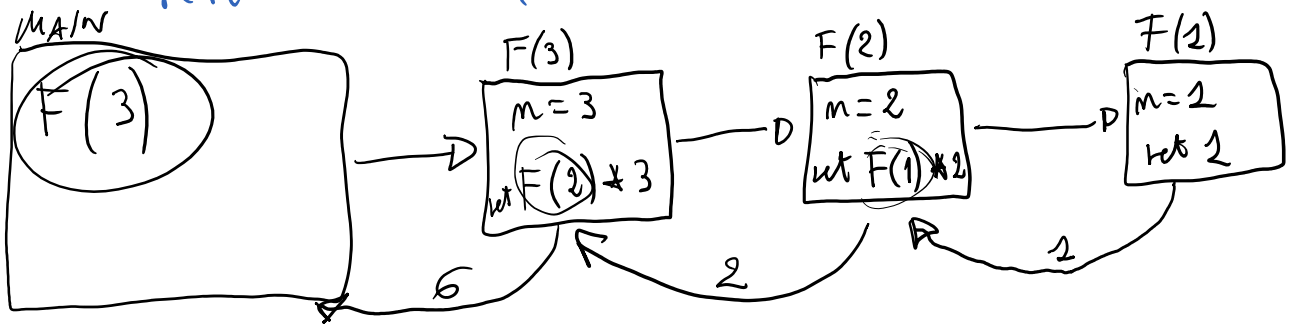
FUNZIONI RICORSIVE

$$F(n) = \underbrace{1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n} \qquad n \geq 1$$

$$\begin{cases} F(n) = F(n-1) \cdot n \\ F(1) = 1 \end{cases}$$

$$F(3) = F(2) \cdot 3 = F(1) \cdot 2 \cdot 3 = 1 \cdot 2 \cdot 3$$

```
def F(n):
    if n == 1:
        return 1
    return F(n-1) * n
```

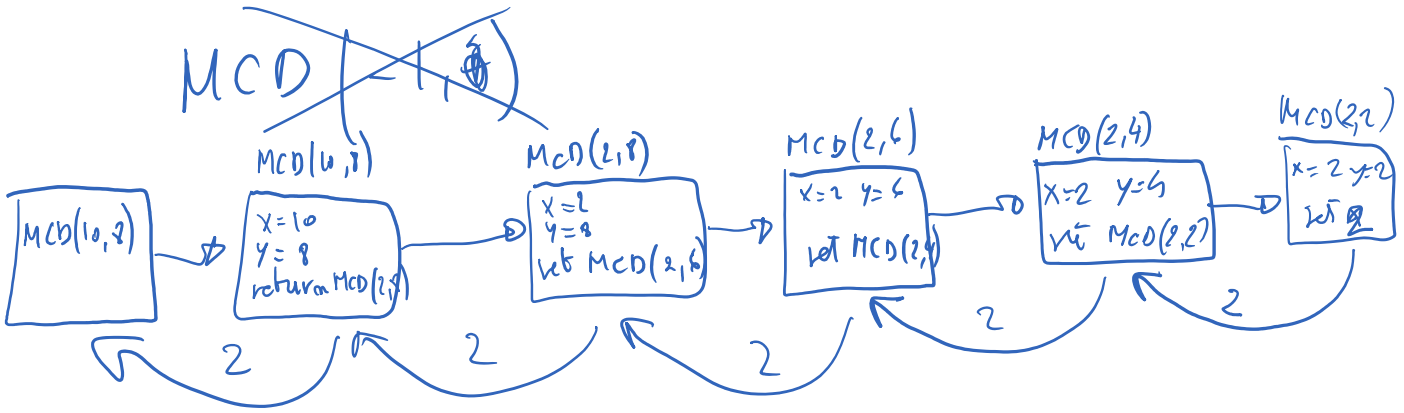



```
def F(n):
    return F(n-1) * n
```

$MCD(x,y)$

- se $x = y \Rightarrow MCD(x,y) = x$
- se $x > y \Rightarrow MCD(x,y) = MCD(x-y, y)$
- se $x < y \Rightarrow MCD(x,y) = MCD(x, y-x)$

```
def MCD(x,y):
    if x==y:
        return x
    if x>y:
        return MCD(x-y,y)
    return MCD(x,y-x)
```



$S(L)$

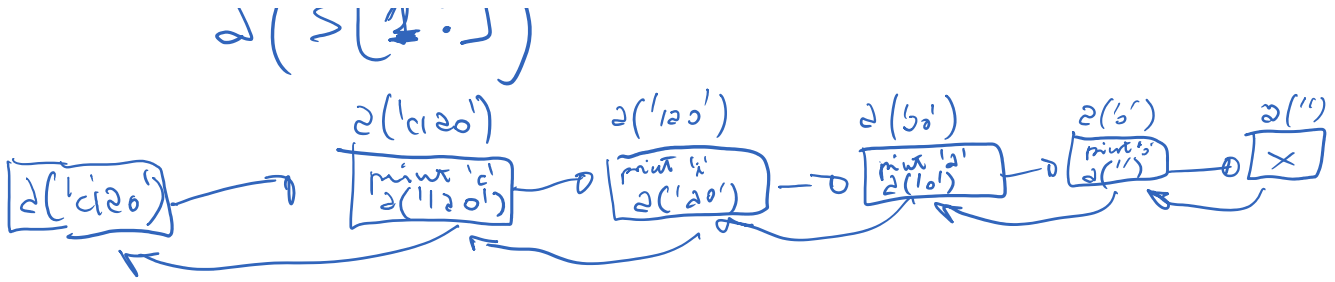
a.b. $L = [] \Rightarrow S(L) = \phi$

c.r. $L = [t \mid L'] \Rightarrow S(L) = t + S(L')$


```
def S(L):
    if not L:
        return ∅
    return L[∅] + S(L[1:])
```

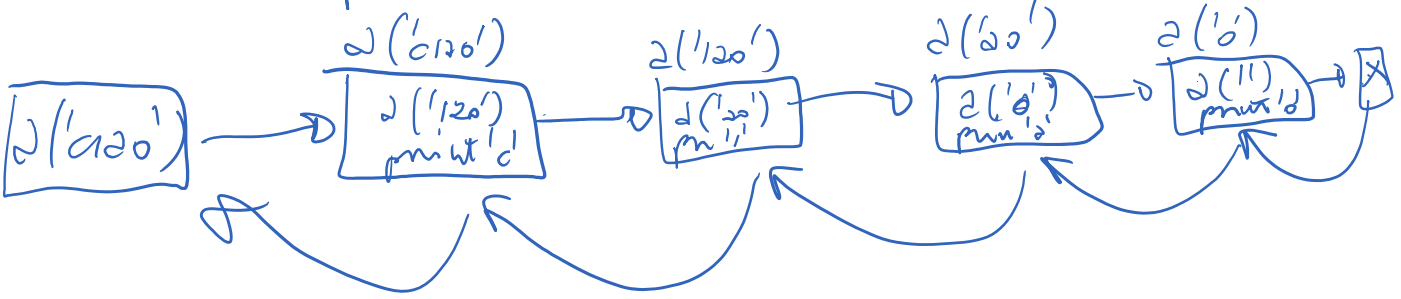
$s = 'ciao'$

```
def a(s):
    if not s:
        return
    print s[∅]
    a(s[1:])
```



Come sopra ma

→ d(s[1:])
print(s[0])



```
def f(n)
    gestione del caso base
    PARTE AVANTI
    CHIAM. RICORSIVA f(n-1)
    PARTE INDIETRO
```

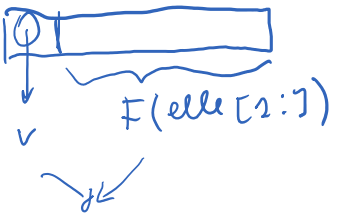
```
d(s)
    g.c.b.
    print s[0]
    d(s[1:])
```

print S[0]

FUNZIONI RICORSIVE - ESERCIZI

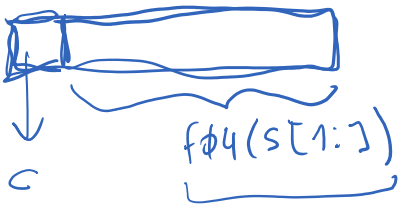
giovedì 6 dicembre 2018 10:32

F.R. che data una lista di interi ella restituisce la somma dei numeri pari in ella.



F.R. DATA UNA STRINGA S restituisco la stringa ottenuta da S eliminando le vocali. f04('ciao') → 'c'

c in "aeiou"



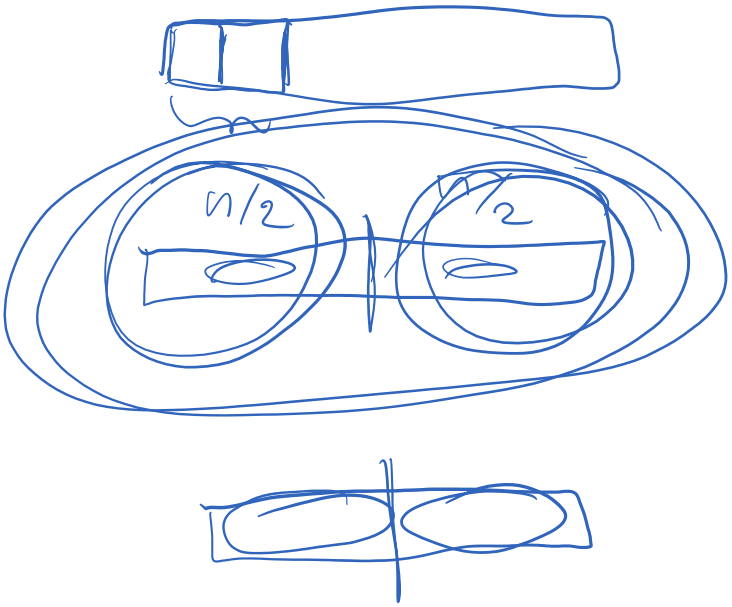
return ('' if s[0] in 'aeiou' else s[0]) + f04(s[1:])

F.R. DATI N1 e N2 interi N1 ≤ N2 restituisce la SOMMA DEI VALORI TRA N1 e N2 COMPRESI !!

F.R. Data stringa S e un carattere c Calcolare # occorrenze di c in S

F.R. s stringa, informa True se S CONTIENE SOLO COPPIE CONSECUTIVE FORMATE DA UNA CIFRA E UN CAR. ALFABETICO.

HINT: t.isdigit() t.isalpha()



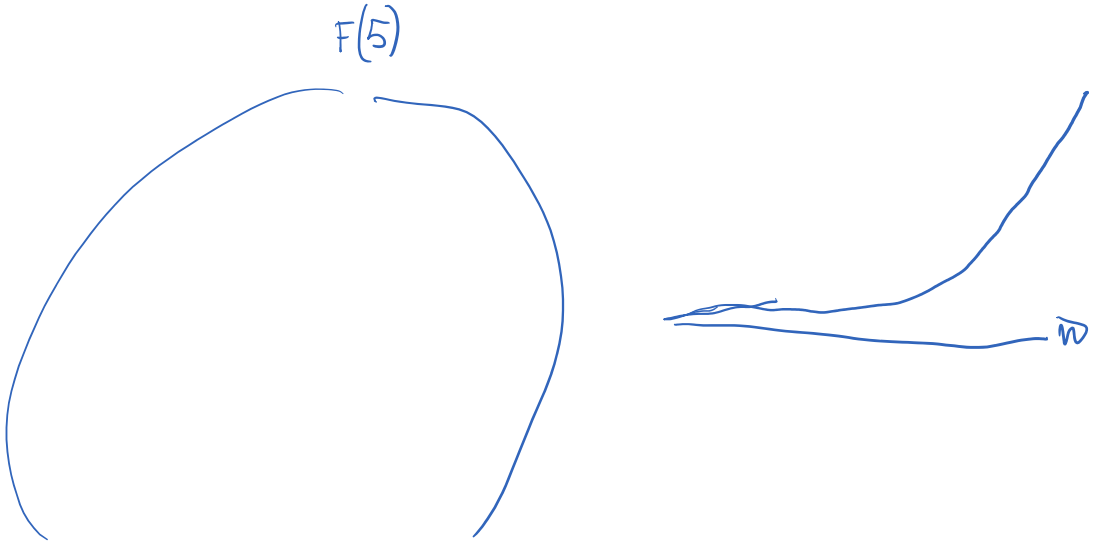
$n = 2^k$

F.R. Intero N

$F(0) = 0$

$F(1) = 1$

$F(n) = F(n-1) + F(n-2)$



CORRETTEZZA DI FUNZ. RICORSIVE

venerdì 7 dicembre 2018 10:57

(1) Nel caso base valgono le bas-condizioni (se vere le pre)

$$PRE(x) \Rightarrow POS(f(1))$$

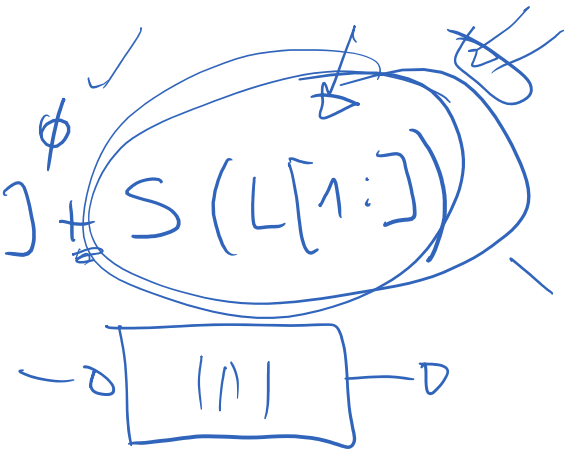
(2) Se vere le pre e non siamo nel caso base ogni chiamata ricorsiva avrà argomenti che soddisfano le PRE

$$PRE(f(x)) \Rightarrow PRE(f(x-1))$$

(3) Non siamo nel caso base.
Assumiamo vere le post sulla ch. ricorsiva $f(x-1)$
Allora saranno vere le post di $f(x)$

$$POS(f(x-1)) \Rightarrow POS(f(x))$$

```
def S(L):
    return ϕ
    return L[ϕ] + S(L[1:])
```

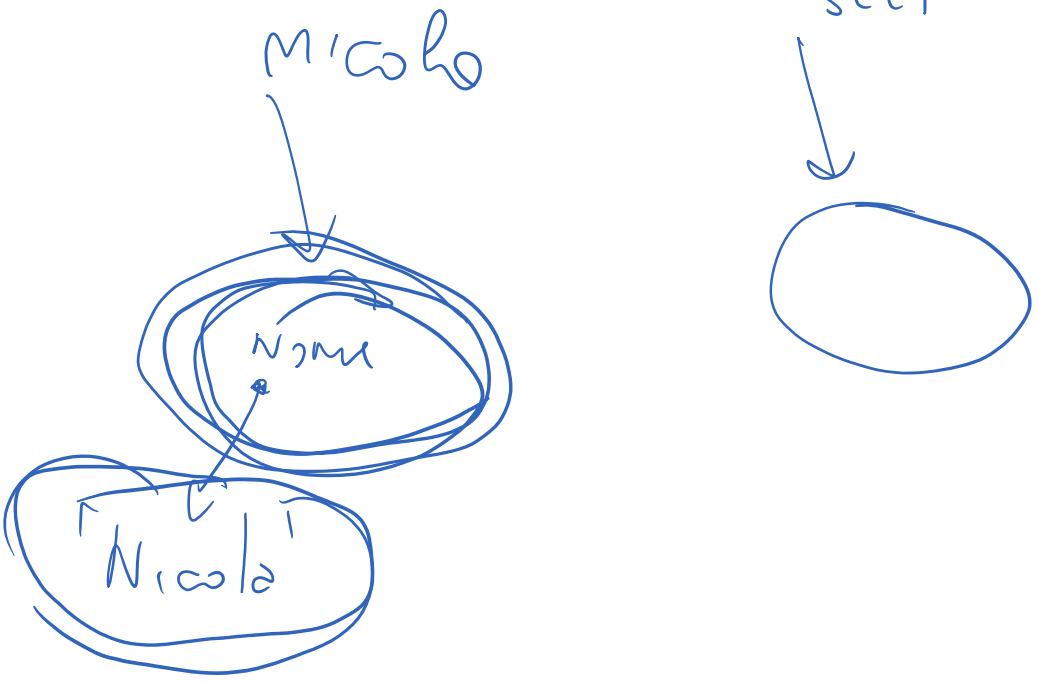


Somma dei numeri in L[1:]

CLASSI

```
class Studente:
    ...
    ...

nicola = Studente()
nicola.nome = 'Nicola'
```


```
def __init__
```



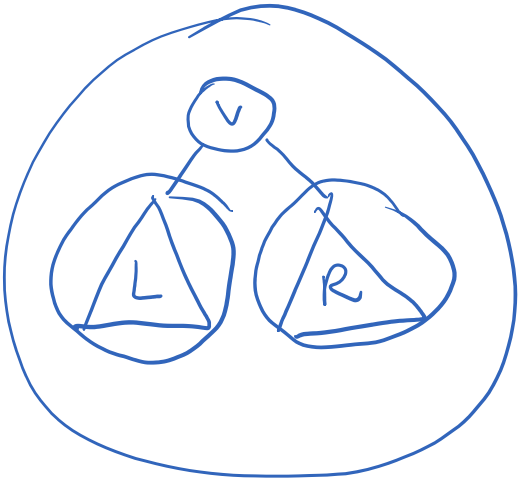
```
class Carta:
    def __init__(self, v, s):
        self.valore = v
        self.seme = s
c = Carta(4, 'fiori')
```

ALBERI BINARI

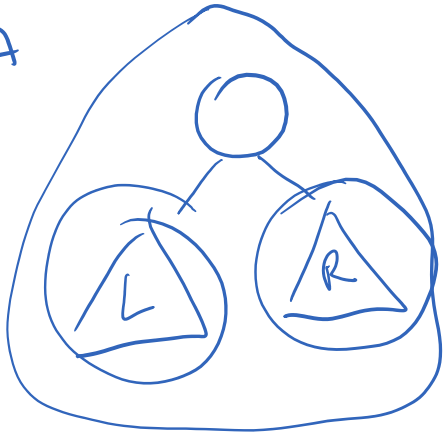
venerdì 14 dicembre 2018 11:00

$$N(-) = \emptyset$$

$$N((r, L, R)) = N(L) + N(R) + 1$$



ALTEZZA



$$A((v, L, R)) = \max(A(L), A(R)) + 1$$

$$\max(h_\phi, h_\phi) + 1 = \phi$$

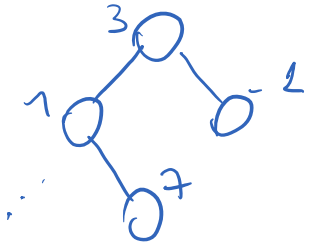
$$t('', s(p, 0))$$

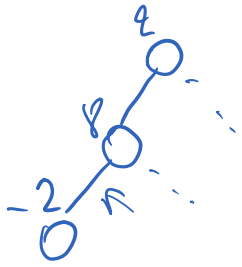
Es. F.R, che dato un albero binario ritorna il num di nodi foglie.

$$F(\_) = \emptyset$$

$$F((v,\_,\_)) = 1$$

$$F((v,L,R)) = F(L) + F(R)$$





$= 3*2 + 1*8$

COMPLESSITÀ ALGORITMICA

mercoledì 19 dicembre 2018 11:30

output modello in

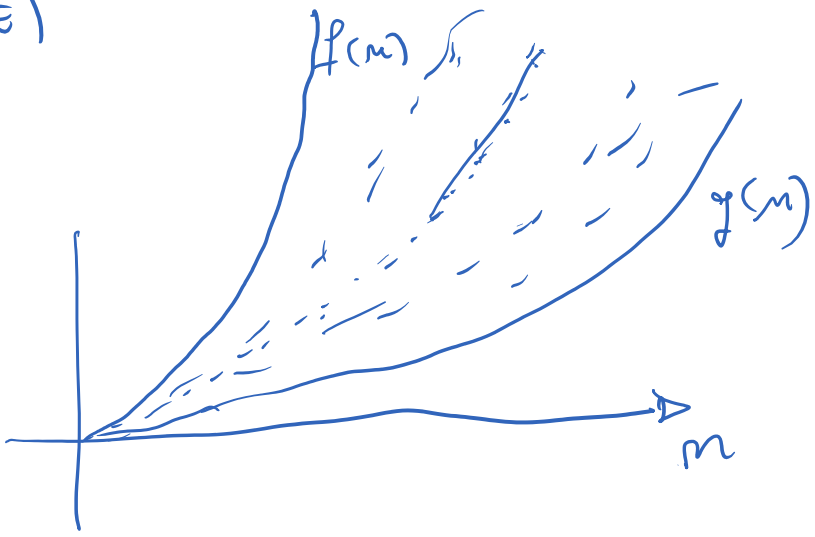
$f(n)$ → $f(n)$ passi

CASO OTTIMO (MIGLIORE)

CASO PESSIMO (PEGGIORE)

CASO MEDIO

- $O(f(n))$
- $\Omega(g(n))$
- $\Theta(h(n))$



1. ASS. COSTO COSTANTE
2. Seq di comandi → costo pari alla somma dei costi dei comandi

3.
```
if COND_A:
    BLOCCO_A
elif COND_B:
    BLOCCO_B
...
else:
```

BLOCCO-ELSE

$$C(\text{IF-ELIF-ELSE}) = C(\text{COND\_A}) + C(\text{COND\_B}) + \dots + MAX(C(\text{BLOCCO\_A}), C(\text{BLOCCO\_B}), \dots)$$

④ while COND:
    BLOCCO

$$C(\text{WHILE}) = m \cdot (C(\text{COND}) + C(\text{BLOCCO}))$$

⑤ for x in IT:
    BLOCCO

$$C(\text{FOR}) = len(IT) \cdot C(\text{BLOCCO})$$

⑥ COSTO FUNZIONI/METODO
COSTO DEL BLOCCO + COSTO CALCOLO ARG.

ALGORITMI DI ORDINAMENTO

giovedì 20 dicembre 2018 10:34

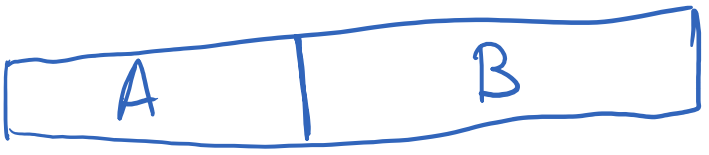
- SELECTION SORT
- INSERTION SORT
- BUBBLE SORT
- MERGE SORT

INTERNI / ESTERNI

IN PLACE

NON IN PLACE

SELECTION SORT



1 3 2 5 7 4

$A = \emptyset$

$B = \{1, \dots, n\}$

$A = [1]$

$A = \{1\}$

$B = \{2, \dots, n\}$

1 2 3 5 7 4

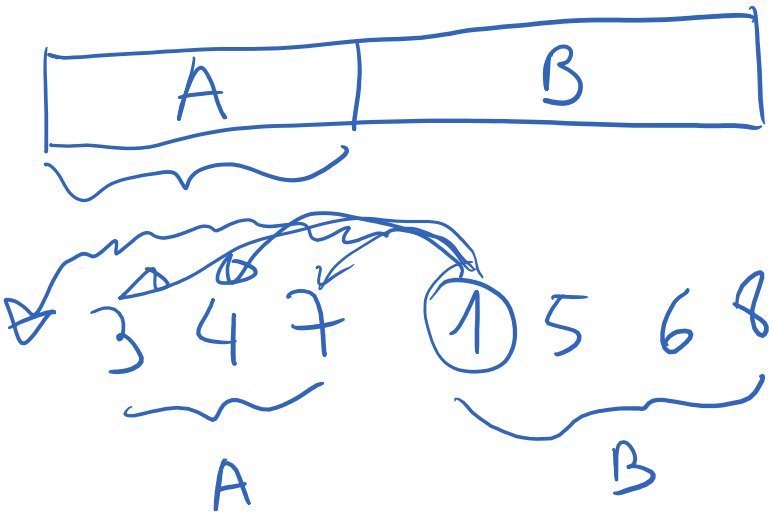
$A = \{1, 2\}$

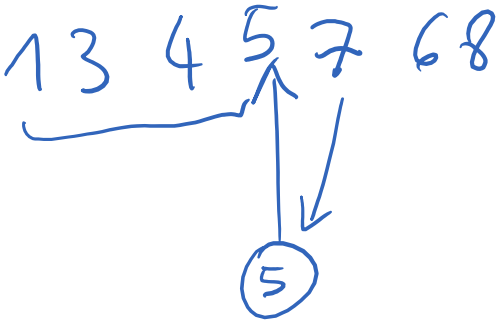
$B = \{3, 4, 5, \dots n\}$

$$\sum_{i=1}^{n-1} (n - i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i$$

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2)$$

INSERTION SORT





$$C.P. \sum_{i=1}^{n-1} i = \frac{n \times (n-1)}{2} = O(n^2)$$

BUBBLE SORT

$$S = [4, 5, 1, 2, 3]$$

1° passo [4, 1, 5, 2, 3]

[4, 1, 2, 5, 3]

[4, 1, 2, 3, 5]

2° passo [1, 4, 2, 3, 5]

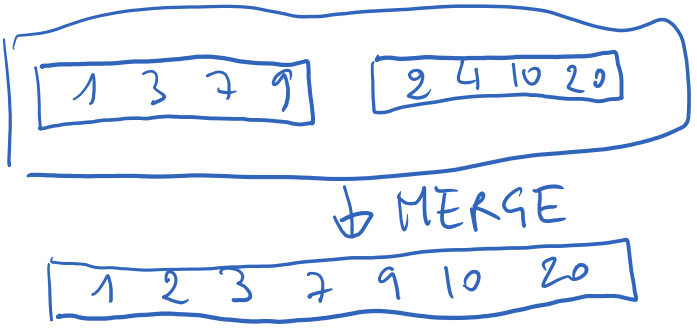
[1, 2, 4, 3, 5]

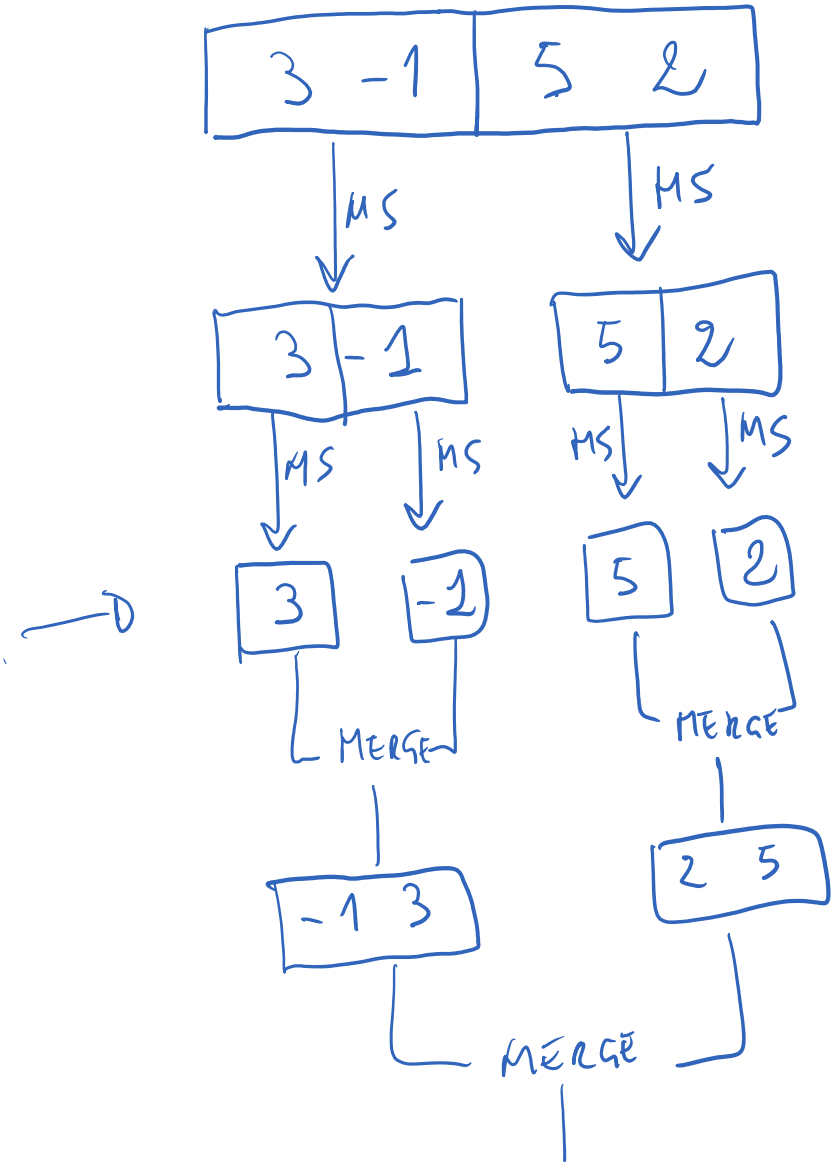
[1, 2, 3, 4, 5]

3° passo

$$\sum_{i} (n-i) = \frac{n(n-1)}{2} = O(n^2)$$

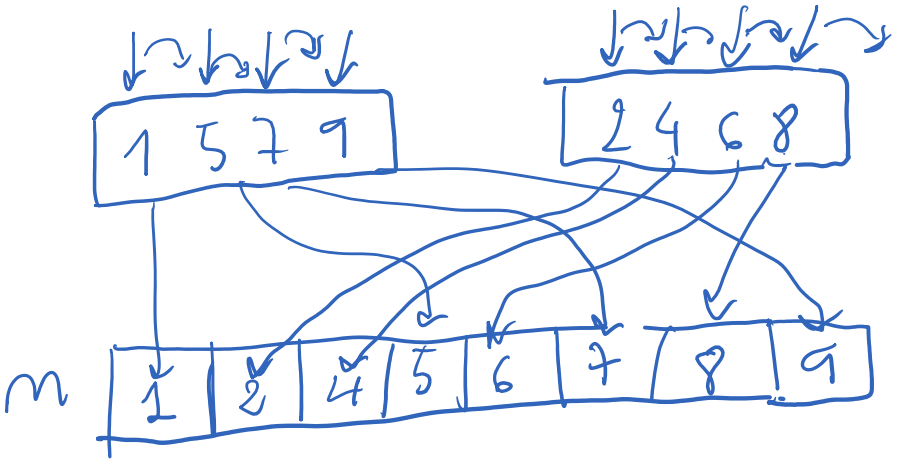
MERGE SORT (DIVIDE ET IMPERA)

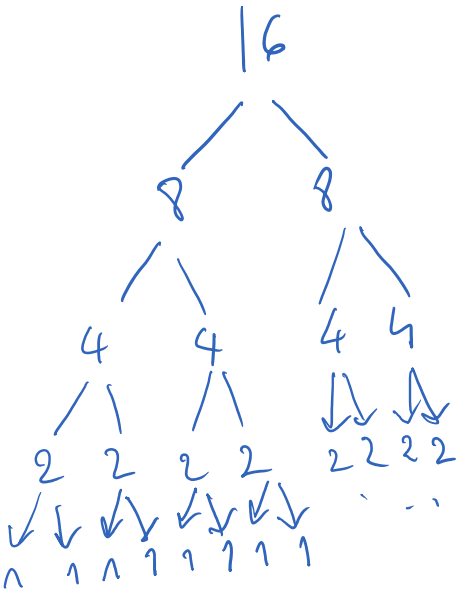




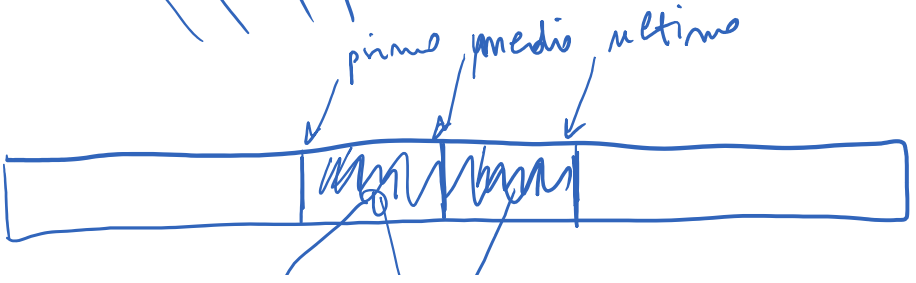
MERGE

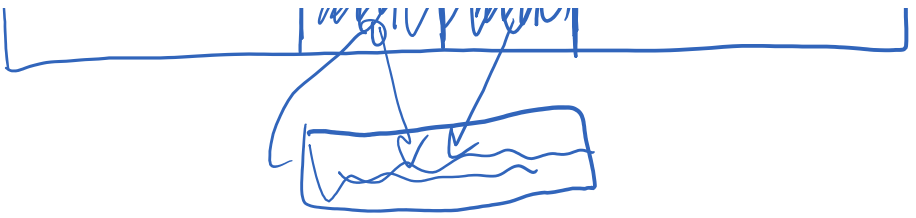
$O(n) = O$





C.C. (TIME) del MergeSort $O(n \log n)$

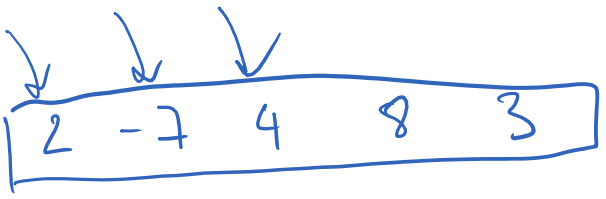


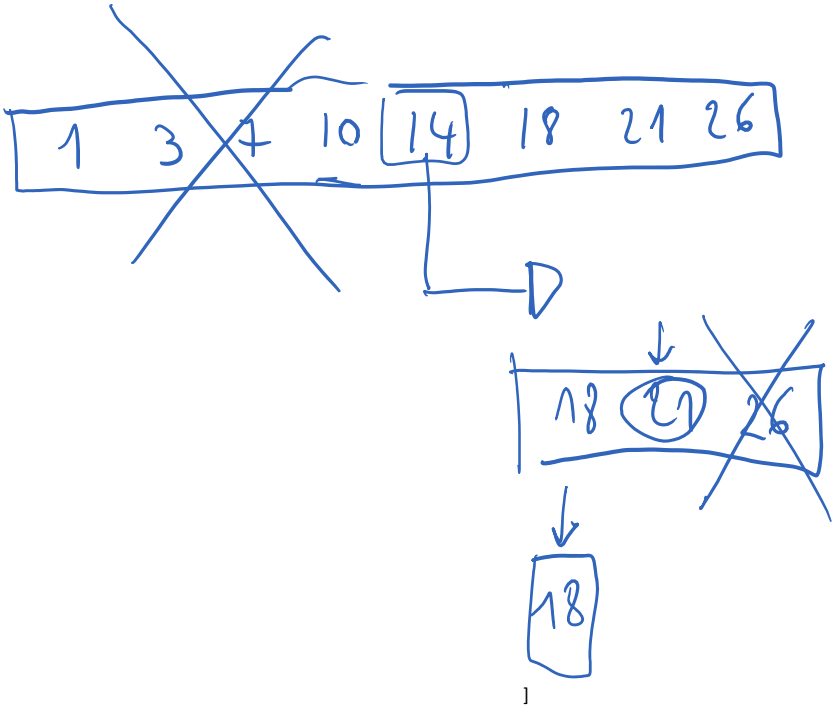


RICERCA

RICERCA LINEARE

$O(n)$





]