Table of Contents

- Quesito con la Susi 1
- Moduli e import
- Contenitori
  - Operazioni sulle liste (list)
  - Altre operazioni sulle liste (list)
  - Operazioni sulle tuple
  - Operazioni sugli insiemi (set)
  - Operazioni sui dizionari (dict)
  - Altre operazioni sui dizionari
- Quesito con la Susi 2
  - Wooclap: secondo voi quante risposte ha azzeccato ?

Ancora argomenti formali delle definizioni delle funzioni
  - Assegnamento multiplo? (WTF?)
  - Ci permette di gestire facilmente gruppi di dati coerenti (struct o record in altri linguaggi)
- "packing" ed "unpacking"
  - E l' "unpacking" cos'è?
  - RIASSUMENDO: se un asterisco precede il nome di una variabile:

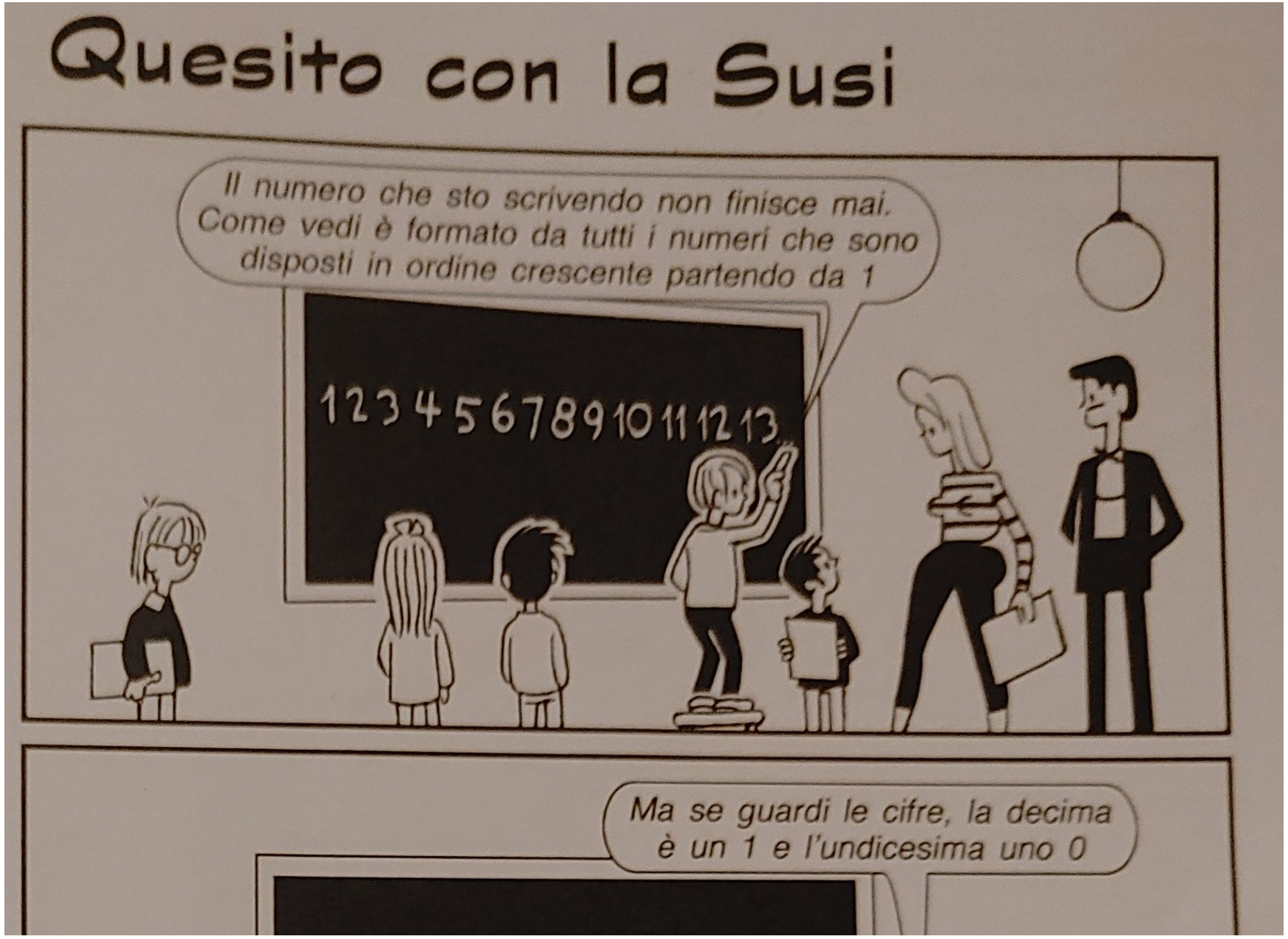
Funzioni ricorsive
- Ogni soluzione ricorsiva ha 4 proprietà
- Wooclap: quanto vale x alla fine dei due programmi ?
- Disegnare alberi (esempio di figura frattale (https://it.wikipedia.org/wiki/Frattale))

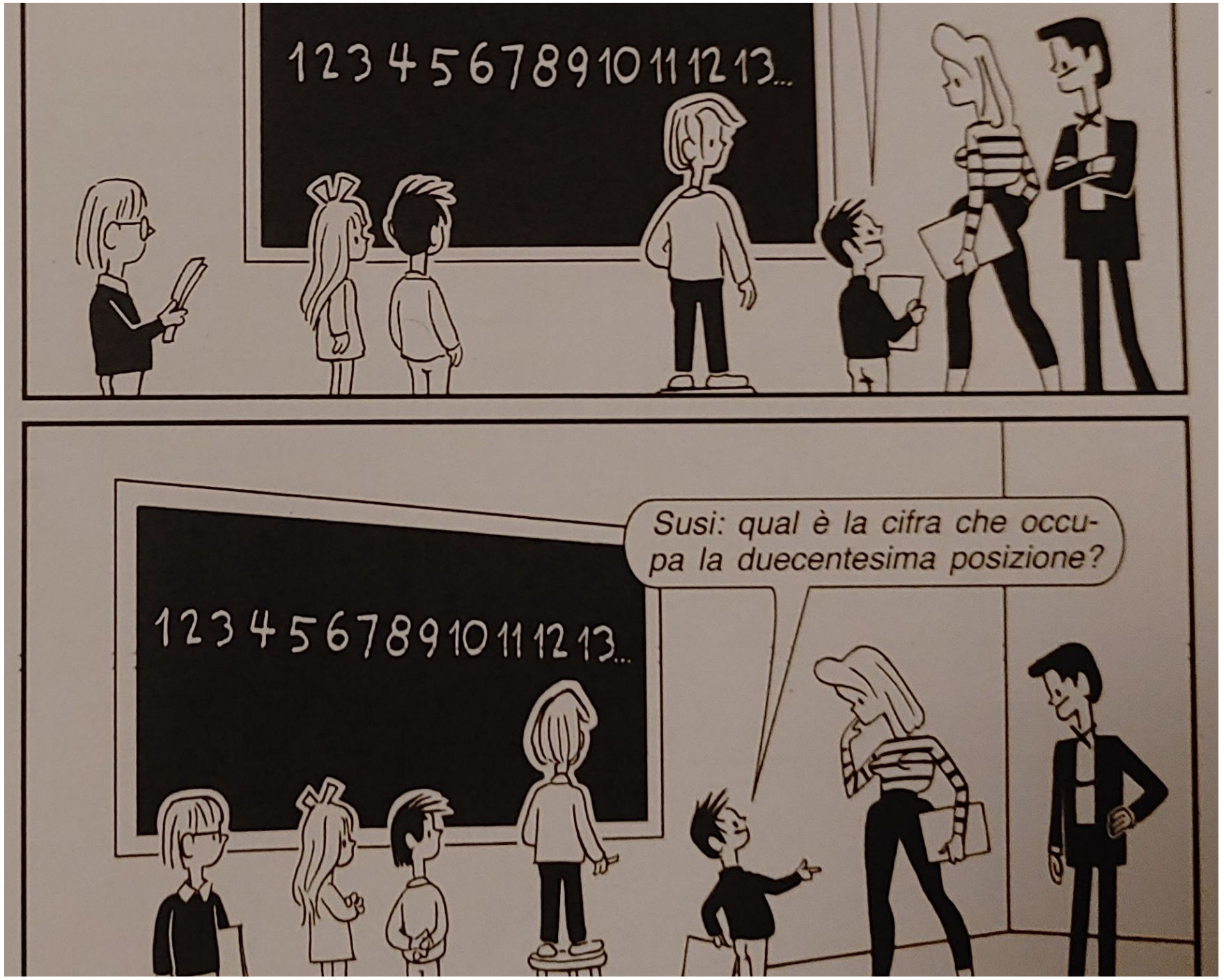
Fondamenti di Programmazione

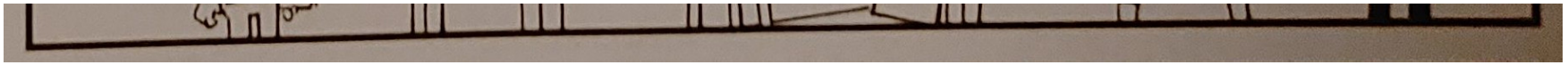
Andrea Sterbini

lezione 5 - 17 ottobre 2022

Quesito con la Susi 1




1 2 3 4 5 6 7 8 9 10 11 12 13...
1 2 3 4 5 6 7 8 9 10 11 12 13...
Susi: qual è la cifra che occupa la duecentesima posizione?



In [5]:

```python
# Per calcolare la 200° cifra:
    # costruisco una stringa con le cifre (fin dove? 200?)
    # estraggo il 200° carattere
def cerca_carattere(N):
    testo = ''
    i = 1
    while len(testo) < N:
        testo += str(i)
        i += 1
    #print(testo)
    return testo[N-1]

# Semplice ma crea tante stringhe
cerca_carattere(2000)
```

Out[5]: '0'

In [20]:

```python
# Seconda idea
# Per calcolare la 200° cifra:
    # trovo in che numero sta la 200° cifra
    # e quante cifre sono rimaste da scandire
    # la tiro fuori
def cerca_senza_stringhe():
    numero, mancanti = cerca_numero()
    print(numero,mancanti)
    testo = str(numero)
    return testo[mancanti-1]

# per trovare in che numero sta la 200° cifra
def cerca_numero():
    # scandisco i numeri da 1 in poi
    cifre = 200
    for i in range(1, 200):
        # calcolo il numero di cifre X del numero corrente
        X = logaritmo_boh(i)
        # se è < del numero di cifre rimaste
        if X < cifre:
            # tolgo X dal numero di cifre ancora da scandire
            cifre -= X
            # incremento il numero I
        # altrimenti
        else:
            # l'ho trovato, lo ritorno
            # assieme al numero di cifre ancora da scandire
            return i, cifre

import math

from math import log10

# per calcolare il numero di cifre di un numero
def logaritmo_boh(N):
    # ne calcolo il logaritmo in base 10
    # lo tronco ad intero
    # gli sommo 1
    return int(math.log10(N))+1

```

```
  41 cerca_senza_stringhe()
```

```
103 2
```

Out[20]: '0'

Moduli e import

Per caricare in memoria funzionalità realizzate in altri file

```
import modulo          # solo 'modulo' viene inserito nel namespace
# chiamare la funzione
modulo.funzione(argomenti)

from modulo import funzione  # solo 'funzione' -> namespace
# chiamare la funzione
funzione(argomenti)
```

Contenitori

Operazioni sulle liste (list)

- **`elemento in L`** True se l'elemento è presente in L
- **`L1 + L2`** nuova lista concatenazione di L1 e L2
- **`L1 * N`** replicazione N volte e concatenazione
- **`L.append(elemento)`** aggiunta di un elemento
- **`L[i]`** elemento all'indice **i** (lettura o assegnamento)
- **`L.pop()`** estrazione distruttiva dell'ultimo elemento (ERR se vuota)
- **`L.pop(i)`** estrazione distruttiva dell'elemento all'indice **i** (ERR se **i** non esiste)
- **`L.insert(i, elemento)`** inserisce l'elemento all'indice i (o in cima o in fondo)

In [31]:
```
# Esempi di operazioni sulle liste
L1 = [ 1, 2, 3, 4, 5, ]
L2 = [ 11, 22, 33, ]
L1 + L2
print(L1.pop(2))
L1.insert(78, 55)
L1
L2[-2:] = []
L2
```

```
3
```

Out[31]:
```
[11]
```

Altre operazioni sulle liste (list)

- **`L.remove(elemento)`** elimina la prima occorrenza dell'elemento (ERR se non presente)
- **`L.index(elemento)`** trova il primo indice in cui c'è l'elemento (ERR se non presente)
- **`L.count(elemento)`** conta l'elemento
- **`L.reverse()`** rovesciamento distruttivo della lista
- **`L.sort()`** ordinamento distruttivo della lista

In [39]:
```
# Altri esempi di operazioni sulle liste
L = [10, 43, 92, 1, -43, 90, -200, int('1') ]
L.count(3)
L.index(-43)
L.sort()
L
```

Out[39]:
```
[-200, -43, 1, 1, 10, 43, 90, 92]
```

Operazioni sulle tuple

- **`elemento in T`** True se l'elemento è presente in T
- **`T1 + T2`** nuova tupla concatenazione di T1 e T2

- **`T * N`** replicazione N volte e concatenazione
- **`T[i]`** accesso all'elemento all'indice i (SOLO lettura)
- **`L.index(elemento)`** trova il primo indice in cui c'è l'elemento (ERR se non presente)
- **`L.count(elemento)`** conta l'elemento

In [41]:
```
# Esempi di operazioni sulle tuple
T1 = 1, 2, 3, 4
T2 = 24, 52, 67, 31
T1 + T2
T1[1:3]
```

Out[41]:
```
(2, 3)
```

Operazioni sugli insiemi (set)

- **`elemento in S`** True se elemento è presente in S
- **`S1 | S2`** unione (or)
- **`S1 & S2`** intersezione (and, elementi in comune)
- **`S1 - S2`** insieme degli el. di S1 che non sono in S2
- **`S1 ^ S2`** insieme degli elementi NON in comune (xor)
- **`S.pop()`** estrazione distruttiva di un el. (ERR if empty)
- **`S.add(elemento)`** aggiunta di un elemento
- **`S.remove(elemento)`** rimozione di un el. (ERR if missing)
- **`S1.update(S2)`** come **`S1 |= S2`** (distruttiva)

In [44]:
```
# Esempi di operazioni sugli insiemi
S1 = set(range(3,101,3)) # insieme dei multipli di 3 in [1..100]
%pprint
S1
```

```
Pretty printing has been turned OFF
```

Out[44]:
```
{3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 6
3, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99}
```

Operazioni sui dizionari (dict)

- **`key in D`** True se la chiave è presente
- **`D[key]`** accesso al valore con chiave key (R/W) (ERR se non presente)
- **`D.keys()`** elenco delle chiavi (che sono uniche)
- **`D.values()`** elenco dei valori (con duplicati)
- **`D.items()`** elenco delle coppie (chiave, valore)
- **`D.popitem()`** torna l'ultima coppia (k,v) e la rimuove
- **`D1 | D2`** nuovo dizionario con tutte le coppie di D1 **e** di D2 (prese nell'ordine)
- **`D1.update(D2)`** modifica D1 aggiungendo le coppie (K,V) di D2 nell'ordine

In [47]:
```
# Esempi di operazioni sui dizionari
D1 = { 1: 'uno', 2: 'due', 3:'tre' }
D2 = { 11: 'undici', 2: 'ventidue', 33:'trentatre' }
D1  | D2
```

Out[47]:
```
{1: 'uno', 2: 'ventidue', 3: 'tre', 11: 'undici', 33: 'trentatre'}
```

Altre operazioni sui dizionari

- **`D.get(key, default)`** se la chiave è presente ne torna il valore, altrimenti torna il valore di default
- **`D.setdefault(key, default)`** se la chiave è presente ne torna il valore altrimenti la inserisce col

valore di default (e lo torna)

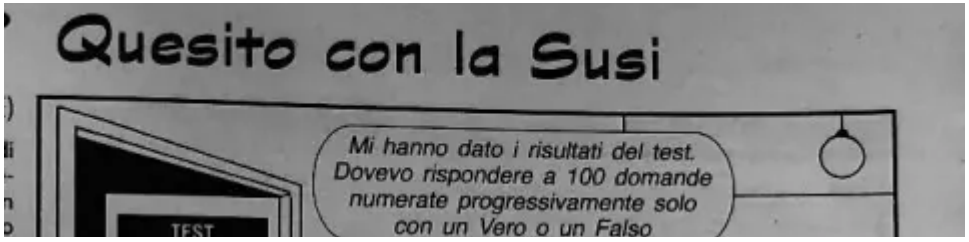
- **`D.pop(key, default)`** se la chiave è presente ne torna il valore e la rimuove, altrimenti torna il valore di default (o ERRORE se no default)
- **`D.fromkeys(keys, value)`** costruisce un dizionario con le chiavi fornite, tutte associate allo stesso valore indicato

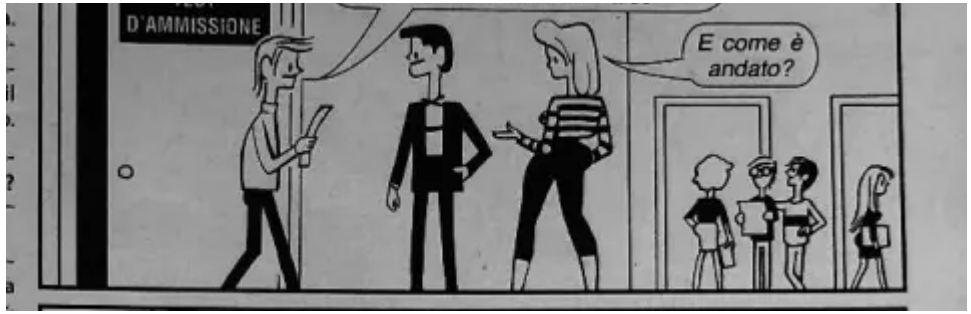
In [50]:
```
# Esempi di operazioni sui dizionari
dict.fromkeys('ABCADEF', 0)
help(dict)
```

```
Help on class dict in module builtins:

class dict(object)
 |  dict() -> new empty dictionary
 |  dict(mapping) -> new dictionary initialized from a mapping object's
 |      (key, value) pairs
 |  dict(iterable) -> new dictionary initialized as if via:
 |      d = {}
 |      for k, v in iterable:
 |          d[k] = v
 |  dict(**kwargs) -> new dictionary initialized with the name=value pairs
 |      in the keyword argument list.  For example:  dict(one=1, two=2)
 |
 |  Built-in subclasses:
 |      StgDict
 |
 |  Methods defined here:
 |
 |  __contains__(self, key, /)
 |      True if the dictionary has the specified key, else False
```

Quesito con la Susi 2





- un amico della Susi ha fatto un esame a crocette True/False con 100 domande
- le risposte giuste **True** erano tutte quelle **multiple di 3** e il resto **False**
- ma lui ha marcato **False** tutte le domande **multiple di 4** ed il resto **True**. Quante ne ha azzeccato?

In [51]:
```python
# per contare le risposte giuste
def conta_azzeccate():
    # inizializzo una variabile conteggio a 0
    azzeccate = 0
    # scandisco i numeri da 1 a 100
    for i in range(1,101):
        # calcoliamo la risposta corretta
        corretta = risposta_corretta(i)
        # calcoliamo la risposta data
        data     = risposta_data(i)
        # se la riposta data è uguale alla risposta corretta
        if corretta == data:
            # incremento il conteggio
            azzeccate += 1
    # alla fine torno il conteggio
    return azzeccate
```

In [52]:
```python
# per calcolare la risposta corretta
def risposta_corretta(X):
    # se il numero è divisibile per 3
        # la risposta corretta è True
    # altrimenti
        # la risposta corretta è False
    return X%3 == 0
```

In [53]:
```python
# per calcolare la risposta data
def risposta_data(X):
    # se il numero è divisibile per 4
        # lui ha risposto False
    # altrimenti
        # lui ha risposto True
    return X % 4 != 0

# Noooo Non vi do subito il risultato 3-)
```

In [56]:
```python
# un altro modo di rispondere: con gli insiemi
# raccolgo le risposte corrette
#   in due insiemi corrette_True e corrette_False
domande       = set(range(1,101))        # i numeri da 1 a 100
corrette_True = set(range(3,101,3))      # i multipli di 3
corrette_False= domande - corrette_True  # i non multipli di 3

# raccolgo le risposte date     True
#   in due insiemi date_True e date_False
date_False = set(range(4,101,4))
date_True = domande - date_False
```

In [58]:
```python
# Le risposte giuste sono:
  # l'intersezione delle risposte date_False con le risposte corrette_False
azzeccate = (corrette_True & date_True) | (corrette_False & date_False)
  # assieme a
  # l'intersezione delle risposte date_True con le risposte corrette_True

# il risultato cercato è il numero di elementi dell'insieme

# Noooo ... non ve lo dico, prima dovete risolverlo voi
```

In [65]:

```python
## Un terzo modo usando i dizionari
# costruisco i dizionari
#     numero -> risposta giusta
#     numero -> risposta data
dizionario_risposte = {}
for i in range(1, 101):
    dizionario_risposte[i] = i % 3 == 0
dizionario_date = {}
for i in range(1, 101):
    dizionario_date[i] = i % 4 != 0
# e poi calcolo l'intersezione dei due insiemi di items
azzeccate_diz = set(dizionario_risposte.items()) & set(dizionario_date.items
# La risposta dopo il break
print(azzeccate)
print(azzeccate_diz)
len(azzeccate_diz)
conta_azzeccate()
```

```
{3, 4, 6, 8, 9, 15, 16, 18, 20, 21, 27, 28, 30, 32, 33, 39, 40, 42, 44, 45, 51,
52, 54, 56, 57, 63, 64, 66, 68, 69, 75, 76, 78, 80, 81, 87, 88, 90, 92, 93, 99,
100}
{(69, True), (4, False), (45, True), (92, False), (27, True), (3, True), (28, Fa
lse), (93, True), (51, True), (8, False), (52, False), (32, False), (76, False),
(18, True), (56, False), (100, False), (80, False), (9, True), (99, True), (75,
True), (42, True), (57, True), (33, True), (66, True), (90, True), (16, False),
(81, True), (6, True), (39, True), (15, True), (40, False), (20, False), (64, Fa
lse), (21, True), (30, True), (44, False), (88, False), (68, False), (87, True),
(63, True), (78, True), (54, True)}
```

Out[65]: 42

Wooclap: secondo voi quante risposte ha azzeccato ?

1 Vai a www.wooclap.com

2 Immettere il codice dell'evento nel banner superiore

Codice evento
F22LEZ5

In [ ]:
```
# vediamo finalmente quante ne ha azzeccate
print('azzeccate', conta_azzeccate())
len(azzeccate)
```

Ancora argomenti formali delle definizioni delle funzioni

Avrete notato che la funzione **print** accetta un numero indefinito di argomenti

Ma come fa? E' speciale?

No, approfitta della sintassi degli **assegnamenti multipli** e del **packing/unpacking**

Assegnamento multiplo? (WTF?)

elenco di variabili = contenitore con stesso numero di elementi

Ciascuna variabile viene assegnata, nell'ordine, con i valori elencati a destra dell' **=**

- **PRIMA** viene calcolato tutto il contenitore a destra
- **POI** viene fatto l'assegnamento

In [1]:
```
A, B = 1, 2
A, B = B, A              # fa uno SCAMBIO!
print(A, B)
```

```
2 1
```

Ci permette di gestire facilmente gruppi di dati coerenti (struct o record in altri linguaggi)

In [3]:
```
# p.es. se abbiamo altezza (H), larghezza (L) e profondità (P)
# di una scatola in quest'ordine in una lista,
dimensioni_scatola = [ 10, 20, 30, ]
# per stamparli dovremmo prima estrarli
H = dimensioni_scatola[0]
L = dimensioni_scatola[1]
P = dimensioni_scatola[2]
# poi stamparli
print('altezza',H)
print('larghezza',L)
print('profondità',P)
# ma che strazio!!!! Bisogna ricordarsi gli indici! (0, 1, 2)
```

```
altezza 10
larghezza 20
profondità 30
```

In [4]:
```
# MOOOLTO meglio!
H, L, P = dimensioni_scatola # li estraggo in ordine e li stampo
print('altezza',H)
print('larghezza',L)
print('profondità',P)
# NOTA: NON dobbiano ricordarci la posizione,
#       basta mettere le variabili nell'ordine giusto
```

```
altezza 10
larghezza 20
profondità 30
```

"packing" ed "unpacking"

In un assegnamento possiamo raccogliere in una sola variabile tutti i valori rimanenti di una lista (**packing**)

Il nome della variabile deve essere preceduto da asterisco *

In [23]:
```
# PACKING: raccolgo in C tutti i valori
# tranne i primi due che vanno in A e B
# mettendo un asterisco subito prima del nome della variabile C
A, B, *C = [ 1, 2, 3, 4, 5, 6]
print('A:',A)
print('B:',B)
print('C:',C)
```

```
A: 1
B: 2
C: [3, 4, 5, 6]
```

In [67]:
```
# Esempio di packing che raccoglie *lista* vuota
A, B, *C = (1,)
print('C ora è', C)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_7645/4150810365.py in <cell line: 2>()
      1 # Esempio di packing che raccoglie *lista* vuota
----> 2 A, B, *C = (1,)
      3 print('C ora è', C)

ValueError: not enough values to unpack (expected at least 2, got 1)
```

In [24]:
```python
# ancora meglio, posso mettere la variabile "packed" all'inizio!!!
*A, B, C = [ 1, 2, 3, 4, 5, 6]
print('A:',A)
print('B:',B)
print('C:',C)
```

```
A: [1, 2, 3, 4]
B: 5
C: 6
```

In [25]:
```python
# oppure in mezzo!!!
A, *B, C = [ 1, 2, 3, 4, 5, 6]
print('A:',A)
print('B:',B)
print('C:',C)
```

```
A: 1
B: [2, 3, 4, 5]
C: 6
```

Basta che ci sia una sola variabile "packed"
in modo che Python sappia cosa mettere nelle altre,
così metterà il resto in quella con asterisco

In [32]:
```python
## Quindi la definizione di print dev'essere simile a

def my_print(*roba_da_stampare, end='\n', sep=' '):
    # roba_da_stampare è una *tupla* con tutti i valori passati
    # NOTA: print prima converte tutto in stringhe
    stringhe = []
    for valore in roba_da_stampare:
        stringhe.append(str(valore))
    # e poi stampa (qui uso print per semplicità)
    print(sep.join(stringhe), end=end)

my_print(1, 1.3, [1, 2, 3])
```

```
1 1.3 [1, 2, 3]
```

E l' "unpacking" cos'è?

Sempre con l'asterisco possiamo "spacchettare" una variabile che contiene più valori all'interno di una altra espressione

```
X = sequenza
[ ..., *X, ... ]
diventa
[ ..., elementi di X, ... ]
```

In [69]:
```
# esempio di unpacking di un *set*
X = {11, 22, 33, 44}
D = 1, 2, *X, 4
print('D ora è', D)
```

```
D ora è (1, 2, 33, 11, 44, 22, 4)
```

In [7]:
```
# Esempio di unpacking di una sequenza di caratteri
X = 'ABCDE'
[ 1, 2, 3, 4, *X, 5, 6, 7, 8, ]    # la espando in mezzo all'altra sequenza
```

Out[7]:
```
[1, 2, 3, 4, 'A', 'B', 'C', 'D', 'E', 5, 6, 7, 8]
```

RIASSUMENDO: se un asterisco precede il nome di una variabile:

- in un **assegnamento** fa il **packing** di zero o più valori in una **lista** (**tupla** se sono gli argomenti di una funzione)
- in una **espressione** fa l'**unpacking** dei valori contenuti nella variabile

Funzioni ricorsive

- le funzioni possono chiamare se stesse

(ogni **chiamata** ha spazio di memoria separato e personale)

- questo è utilissimo per alcuni tipi di problemi
  (lo vedremo bene nella seconda metà del corso)

Esempio classico:

```
def fattoriale(N):
    if N < 2:
        return 1
    else:
        return N * fattoriale(N-1)
```

N	**1**	**2**	**3**	**4**	**5**	**6**	**...**
F(N)	1	2	6	24	120	720	...

In [70]:

```
from rtrace import trace  # per tracciare l'esecuzione
# la "annoto" in modo che si fermi ad ogni chiamata ed uscita
@trace(pause=True)
def fattoriale(N):
    if N < 2:
        return 1          # caso base (conosciuto)
    else:
        return N * fattoriale(N-1) # caso ricorsivo (riduzione e ricomposizi

# vediamo che fa
print(fattoriale(5))      # vediamo il risultato
```

```
120
```

In [71]:
```
1 fattoriale.trace(5)       # vediamo la traccia delle chiamate
```

```
------------------- Starting recursion -------------------
 entering       fattoriale(5,)
|-- entering    fattoriale(4,)
|--|-- entering fattoriale(3,)
|--|--|-- entering      fattoriale(2,)
|--|--|--|-- entering   fattoriale(1,)
|--|--|--|-- exiting    fattoriale(1,)  returns 1
|--|--|-- exiting       fattoriale(2,)  returns 2
|--|-- exiting  fattoriale(3,)  returns 6
|-- exiting     fattoriale(4,)  returns 24
 exiting        fattoriale(5,)  returns 120
-------------------- Ending recursion --------------------
Num calls: 5
```

Out[71]:
```
120
```

Ogni soluzione ricorsiva ha 4 proprietà

- esiste almeno una soluzione conosciuta (**caso base**)

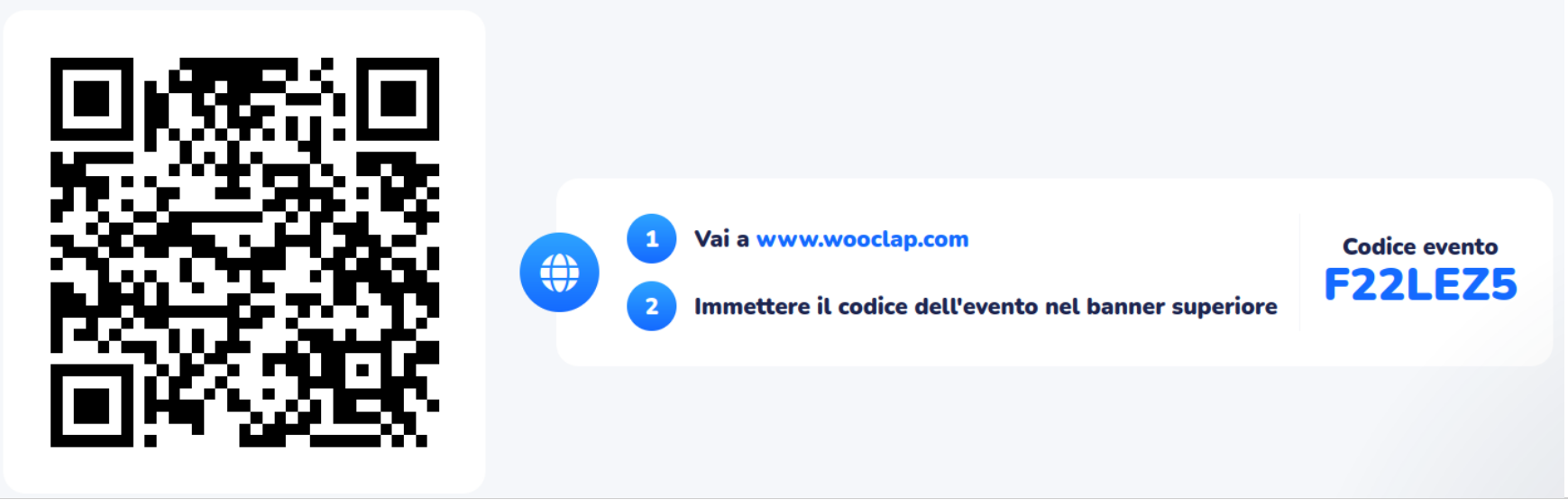
- il problema può essere decomposto in sottoproblemi "più piccoli" (**riduzione**)

- a forza di ridurre i problemi si casca sempre in uno dei casi base (**convergenza**)

- dalle soluzioni dei sottoproblemi possiamo costruire la soluzione del problema (**ricomposizione**)

Questo schema ci può aiutare a trovare una soluzione ricorsiva ad un nuovo problema (seconda metà del corso)

Wooclap: quanto vale `x` alla fine dei due programmi ?



In [72]:
```python
# soluzioni: 0, 1, 3, 6, *10*, ...
def increase(x):
    if x == 0:
        return 0           # caso base
    x += increase(x-1)     # chiamata ricorsiva
    return x
x = increase(4)
x
# non è molto diversa dal fattoriale, ma fa la somma
```

Out[72]: 10

In [73]:
```
# soluzione
x = 4
increase(x)
increase(x)
x
# la variabile x non viene modificata
# anche se si chiama come l'argomento x
```

Out[73]: 4

Disegnare alberi (esempio di figura frattale (https://it.wikipedia.org/wiki/Frattale))

- Un albero di livello 0 è una foglia (cerchietto) (caso base)
- Un albero di livello N è formato da: (ricomposizione)
  - un tronco lungo X
  - un **albero di livello N-1** inclinato a sinistra, con tronco 80% di X (sottoproblema)
  - un **albero di livello N-1** inclinato a destra, con tronco 70% di X (sottoproblema)

Questa è una definizione **ricorsiva**: i due rami sono due alberi!

Mi serve uno strumento di disegno:

- col modulo **turtle** posso tracciare movimenti relativi alla tartaruga
- col modulo **random** posso generare colori casuali

In [74]:

```
# Per disegnare un albero frattale serve una tartaruga e dei numeri casuali
import turtle
from random import randint      # generatore di interi casuali
turtle.colormode(255)           # setto i colori in modalità RGB
t = turtle.Turtle()             # creo una tartaruga
t.penup()                       # alzo la penna
t.left(90)                      # giro verso l'alto
t.back(200)                     # mi posiziono in basso
t.pensize(5)                    # con penna cicciotta
t.speed(0)                      # e velocità alta
#help(turtle)                   # see also      (oppure 'pydoc turtle')
```

In [75]:

```
# cos'è un albero frattale?
# ha un tronco che regge due rami
# i rami sono inclinati a sinistra e a destra del tronco
# ciascun ramo ha la stressa struttura di un albero
# ma leggermente più piccolo
# un albero di livello zero è una foglia
def albero(t, tronco, angolo, livelli):
    'disegno un albero con un certo tronco iniziale e # di livelli'
    # TODO
    if livelli == 0:
        draw_leaf(t)
    else:
        # disegno il tronco (e mi sposto alla fine)
        draw_trunk(t, tronco)
        # mi giro a sinistra
        t.left(angolo)
        # disegno il ramo sinistro, più piccolo 80%
        albero(t, tronco * 0.8, angolo, livelli-1)
        # mi giro a destra
        t.right(angolo*2)
        # disegno il ramo destro, più piccolo 70%
        albero(t, tronco * 0.7, angolo, livelli-1)
        # torno nella direzione iniziale
        t.left(angolo)
        # torno alla base del tronco
        t.back(tronco)
```

In [77]:
```
# devo definire come disegnare il tronco e la foglia
def draw_trunk(t, lunghezza):
    'Disegno un tratto di colore casuale'
    # TODO
    # cambio colore a caso
    R = randint(100, 200)
    G = randint(100, 200)
    B = randint(100, 200)
    t.color(R,G,B)
    # abbasso la penna
    t.pendown()
    # mi muovo in avanti di lunghezza pixel
    t.forward(lunghezza)
    # alzo la penna
    t.penup()
    # NOTA: ora sono all'estremo opposto del tronco
```

In [78]:
```
def draw_leaf(t):
    'disegno una foglia col colore corrente'
    # TODO
    # abbasso la penna
    t.pendown()
    # disegno un pallino
    t.dot()
    # alzo la penna
    t.penup()



```

In [79]:
```
## Vediamo se funziona :-)
t.clear() # pulisco il foglio
albero(t,100,30,6)
```