Table of Contents

Fondamenti di Programmazione

Andrea Sterbini

lezione 16 - 28 novembre 2022

In [32]:
```
1 %load_ext nb_mypy
```

```
Version 1.0.4
```

RECAP:

- Colori come oggetti (con matematica dei colori)
- Immagini come matrici di Colori
- Filtri come oggetti che generano il colore del nuovo pixel
- Figure geometriche

METODOLOGIA di analisi Object Oriented

- Si individuano le strutture dati necessarie (**class**)
- con i loro **attributi** (informazioni "personali" di ciascun dato)
  - se una informazione è identica e comune a tutti gli individui la posso mettere come **attributo di classe**
- la loro inizializzazione (`__init__`)
- i **metodi** fondamentali (operazione sui dati o che calcolano valori a partire dagli attributi)
  - se vogliamo creare l'oggetto in altri modi possiamo creare un **@classmethod**

Ogni volta che vogliamo aggiungere una funzionalità ci dobbiamo chiedere quale oggetto deve essere "responsabile" (oppure "conosce le informazioni") per calcolarla

Metafora dell'ufficio

- E' un po' come voler organizzare un ufficio con persone che hanno **tipi di mansioni** diverse
  - ciascuno ha le sue informazioni e si cerca di non assegnarle a più uffici (classi)
  - gli scambi tra impiegati devono essere minimi

Ereditarietà (Specializzazione/ Riuso e Ampliamento delle funzionalita)

Se si vede che diverse classi condividono dei comportamenti/attributi comuni

- definiamo una super-classe che contiene l'implementazione comune dei metodi o gli attributi comuni
- nelle sottoclassi che ereditano da questa mettiamo solo gli attributi diversi ed i metodi diversi (se necessario c'è anche il modo di chiamare i metodi della superclasse)

Esempio: tutti gli animali visualizzati in un gioco hanno una posizione, una icona, un verso ... l'insieme degli attributi e

metodi comuni può essere messo nella classe Animale da cui ereditano Cane, Gatto, Cavallo, Ornitorinco

In [46]:

```
# Tutte le Figure hanno:
# - posizione
# - colore
# - un metodo che le disegna (che per default non fa nulla)
# - un metodo che ne calcola l'area (per default 0)

# Ciascuna specifica figura
# - ha dei parametri specifici (raggio, lati, cateti, fuochi, retta e fuoco, ...)
# - ha il suo metodo specifico che la disegna
# - ha il suo metodo specifico che ne calcola l'area

# definiamo la gerarchia di classi:
# Figura
#   Punto
#   Linea
#       Freccia
#   Rettangolo
#       Quadrato
#       TriangoloRettangolo
#   PoligonoRegolare
#       Triangolo
#       Pentagono
#   EllisseApprossimataDaCerchi
#       Cerchio
import turtle
t = turtle.Turtle()
turtle.colormode(255)
```

In [47]:

```
class Figura:
    def __init__(self, x, y, colore, direzione=0, spessore=1, campitura=None):
        self._x = x
        self._y = y
        self._colore = colore
        self._direzione = direzione
        self._spessore = spessore
        self._campitura = campitura

    def area(self):
        raise Exception("Il metodo 'area' non è stato implementato")

    def draw(self, turtle):
        turtle.end_fill()
        turtle.up()
        turtle.goto(self._x,self._y)
        turtle.setheading(self._direzione)
        turtle.color(self._colore)
        turtle.pensize(self._spessore)
        turtle.down()
        if self._campitura:
            turtle.fillcolor(self._campitura)
            turtle.begin_fill()

    def move(self, x, y):
        self._x = x
        self._y = y
```

In [61]:

```
class Rettangolo(Figura):
    def __init__(self, x, y, colore, larghezza, altezza,
                 direzione=0, spessore=1, campitura=None):
        "creo un nuovo rettangolo"
        # riuso l'inizializzazione della mia superclasse per
        # gli attributi che già sa gestire
        super().__init__(x, y, colore, direzione, spessore, campitura)
        # gestisco qui gli attributi aggiuntivi
        self._larghezza = larghezza
        self._altezza = altezza

    def area(self):
        "calcolo l'area del rettangolo"
        return self._larghezza * self._altezza

    def draw(self, turtle):
        """
        Disegno il rettangolo son la tartaruga fornita come parametro.
        """
        super().draw(turtle)
        turtle.forward(self._larghezza)
        turtle.left(90)
        turtle.forward(self._altezza)
        turtle.left(90)
        turtle.forward(self._larghezza)
        turtle.left(90)
        turtle.forward(self._altezza)
        turtle.left(90)
        turtle.end_fill()
```

In [62]:

```
class PoligonoRegolare(Figura):
    "Costruisco un poligono regolare"
    def __init__(self, x, y, lato, N, colore,
                 direzione=0, spessore=1, campitura=None):
        super().__init__(x, y, colore, direzione, spessore, campitura)
        self._N = N
        self._lato = lato

    def draw(self, turtle):
        super().draw(turtle)
        angolo_esterno = 360/self._N
        for _ in range(self._N):
            turtle.forward(self._lato)
            turtle.left(angolo_esterno)
        turtle.end_fill()


class Quadrato(Rettangolo):
    def __init__(self, x, y, lato, colore,
                 direzione=0, spessore=1, campitura=None):
        super().__init__(x, y, colore, lato, lato,
                         direzione, spessore, campitura )
```

In [63]:

```
r = Rettangolo(50, 100, (255,0,0), 200, 100,
               direzione=45, spessore=5,
               campitura=(255,255,0))

p = PoligonoRegolare(-100, -100, 50, 5, (0,0,255),
                     20, 4, campitura=(255,0,0) )

q = Quadrato(100, 100, 90, (0,255,0), -30, 10)

r.draw(t)

r.move(-200,-100)

r.draw(t)

p.draw(t)

q.draw(t)

print(r.area(), q.area())


```

```
20000 8100
```

NUOVO ARGOMENTO: Problemi e soluzioni ricorsive

Una funzione è ricorsiva se chiama se' stessa (principio di induzione).

Un problema ammette una soluzione ricorsiva se:

- è possibile rimpicciolire la dimensione del problema da risolvere (**riduzione**)
- esiste almeno un problema che ha una soluzione elementare (**caso base**)
- è sempre possibile, applicando ripetutamente la riduzione, arrivare ad un caso base (**convergenza**)
- è possibile ottenere la soluzione del problema iniziale dalle soluzioni dei sottoproblemi (**composizione**)

Esempio classico: il fattoriale

DEF: il fattoriale di un numero N intero positivo è il prodotto dei numeri da 1 a N

- come ridurre il problema? **diminuisco N di 1**
- quale soluzione semplice conosco? **F(1) = 1**
- come ottengo F(N) da F(N-1)? **lo moltiplico per N**
- il passo di riduzione converge al caso base? **Sì, sottraendo 1 a N alla fine si arriva ad 1**

Esempio classico: fattoriale(N)

proprietà	esempio
caso base	`F(1) = 1`
riduzione	`F(N) -> F(N-1)`
convergenza	`N, N-1, ..... 1`
composizione	`F(N) = N*F(N-1)`

Schema di implementazione ricorsiva

```
def funzione(problema):
    if is_caso_base(problema):
        return soluzione nota
    else:
        sottoproblema  = riduzione(problema)
        sottosoluzione = funzione(sottoproblema)
        soluzione      = composizione(sottosoluzione)
        return soluzione
```

In [64]:

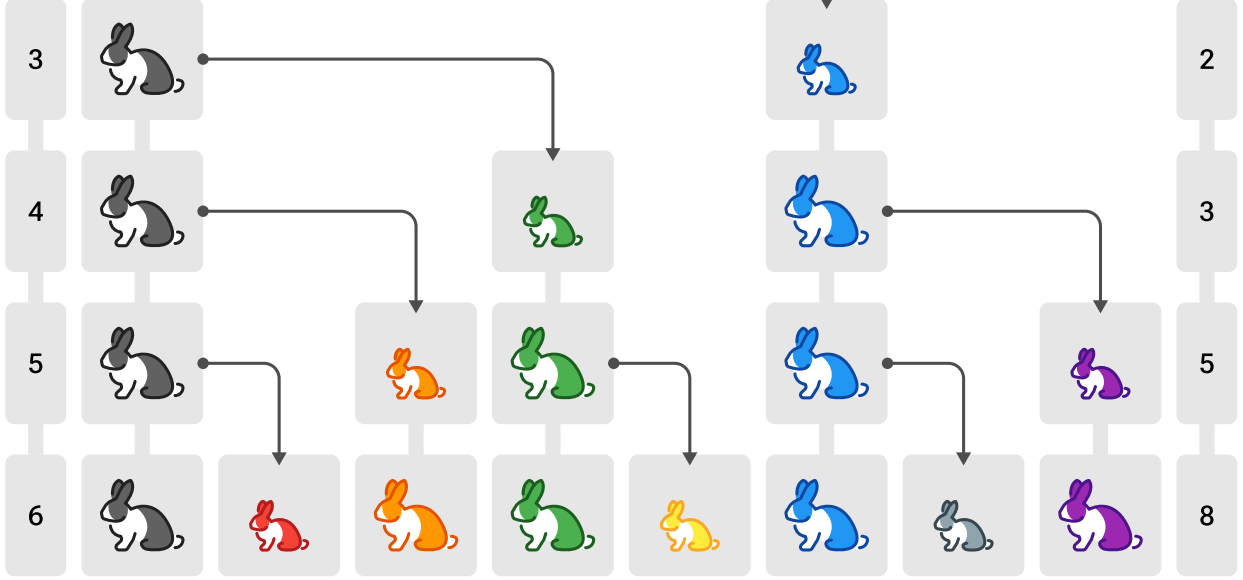
```python
from rtrace import trace

@trace
def fattoriale(N : int) -> int :
    if N==1:
        return 1
    else:
        return N*fattoriale(N-1)

fattoriale.trace(5)
```

```
------------------- 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[64]: 120

Altro caso classico: i coniglietti di Fibonacci

- Una coppia di conigli il primo mese è giovane e non prolifica
- dal secondo mese in poi fa una coppia di coniglietti ogni mese



Ad ogni mese il numero di coppie si ottiene sommando:

- i nuovi coniglietti che nascono dalle coppie adulte (quelle di 2 mesi prima)
- e le coppie correnti (1 mese prima)

Soluzione iterativa (simulando le coppie)

- F(0)=0
- F(1)=1
- ...
- F(N)=F(N-1)+F(N-2)

Soluzione ricorsiva

Se osserviamo il problema dal punto di vista dell'anno N:

- **caso base**: 0 coppia se N==0
- **caso base**: 1 coppia se N==1 (erano troppo giovani)
- **riduzione del problema**: Per calcolare F(N) ci servono F(N-1) ed F(N-2)
- **composizione della soluzione**: F(N) = F(N-1) + F(N-2)

In [65]:

```
@trace
def fibonacci(N : int) -> int :
    if N < 2:
        return 1
    else:
        return fibonacci(N-1) + fibonacci(N-2)

fibonacci.trace(4)
```

```
------------------- Starting recursion -------------------
 entering       fibonacci(4,)
|-- entering    fibonacci(3,)
|--|-- entering fibonacci(2,)
|--|--|-- entering      fibonacci(1,)
|--|--|-- exiting       fibonacci(1,)   returns 1
|--|--|-- entering      fibonacci(0,)
|--|--|-- exiting       fibonacci(0,)   returns 1
|--|-- exiting  fibonacci(2,)   returns 2
|--|-- entering fibonacci(1,)
|--|-- exiting  fibonacci(1,)   returns 1
|-- exiting     fibonacci(3,)   returns 3
|-- entering    fibonacci(2,)
|--|-- entering fibonacci(1,)
|--|-- exiting  fibonacci(1,)   returns 1
|--|-- entering fibonacci(0,)
|--|-- exiting  fibonacci(0,)   returns 1
|-- exiting     fibonacci(2,)   returns 2
 exiting        fibonacci(4,)   returns 5
-------------------- Ending recursion --------------------
Num calls: 9
```

Out[65]: 5

METODO: le 4 proprietà vi guidano per affrontare un problema ricorsivo sconosciuto:

- trovate il **caso base**
- confrontate **problemi** di dimensioni diverse per capire come fare la "riduzione"
- confrontate **soluzioni** di dimensioni diverse per capire come calcolare la "soluzione"
- **verificate che** applicando più volte il passo di riduzione **si arrivi sempre ad un caso base** (convergenza)
- altrimenti aumentate i casi base

E' sempre possibile simulare un ciclo con una ricorsione

E' sempre possibile simulare una ricorsione con uno o più cicli (e se necessario una pila/stack)

- Iacopini Bohm

In [67]:
```python
#  0 1 1 2 3 5 8 13 21

def fibonacci_iter(N : int) -> int :
    coppie = [0, 1]
    for i in range(N):
        coppie.append(coppie[-1] + coppie[-2])
    print(coppie)
    return coppie[-1]

print(fibonacci_iter(7))
# MA ATTENZIONE!!!
# E' sufficiente ricordasi SOLO dei DUE mesi precedenti!
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21]
21
```

In [70]:

```python
# Mi ricordo SOLO dei DUE mesi precedenti
def fibonacci_iter2(N : int) -> int :
    corrente, precedente = 1, 0
    for i in range(N):
        corrente, precedente = corrente+precedente, corrente
    return corrente

print(fibonacci_iter2(70))
```

```
308061521170129
```

In [72]:

```python
# simuliamo l'allevamento anno per anno da 1 ad N con la ricorsione
# - convertiamo le variabili di stato del ciclo in argomenti della funzione
@trace
def _fibonacci_ric_efficiente(N : int, i : int,
                              corrente : int,
                            precedente : int):
    if i == N:
        return corrente
    else:
        corrente, precedente = corrente+precedente, corrente
        return _fibonacci_ric_efficiente(N, i+1, corrente, precedente)

# definiamo una funzione di appoggio che inizializza le variabili di stato
def fibonacci_ric_efficiente(N : int):
    return _fibonacci_ric_efficiente(N, 0, 1, 0)

# oppure potevamo dare dei valori di default appropriati alle "variabili di stato"

_fibonacci_ric_efficiente.trace(7, 0, 1, 0)
```

```
------------------- Starting recursion -------------------
 entering       _fibonacci_ric_efficiente(7, 0, 1, 0)
|-- entering    _fibonacci_ric_efficiente(7, 1, 1, 1)
|--|-- entering _fibonacci_ric_efficiente(7, 2, 2, 1)
|--|--|-- entering      _fibonacci_ric_efficiente(7, 3, 3, 2)
|--|--|--|-- entering   _fibonacci_ric_efficiente(7, 4, 5, 3)
|--|--|--|--|-- entering        _fibonacci_ric_efficiente(7, 5, 8, 5)
|--|--|--|--|--|-- entering     _fibonacci_ric_efficiente(7, 6, 13, 8)
|--|--|--|--|--|--|-- entering  _fibonacci_ric_efficiente(7, 7, 21, 13)
|--|--|--|--|--|--|-- exiting   _fibonacci_ric_efficiente(7, 7, 21, 13) returns 21
|--|--|--|--|--|-- exiting      _fibonacci_ric_efficiente(7, 6, 13, 8)  returns 21
|--|--|--|--|-- exiting         _fibonacci_ric_efficiente(7, 5, 8, 5)   returns 21
|--|--|--|-- exiting    _fibonacci_ric_efficiente(7, 4, 5, 3)    returns 21
|--|--|-- exiting       _fibonacci_ric_efficiente(7, 3, 3, 2)    returns 21
|--|-- exiting  _fibonacci_ric_efficiente(7, 2, 2, 1)    returns 21
|-- exiting     _fibonacci_ric_efficiente(7, 1, 1, 1)    returns 21
 exiting        _fibonacci_ric_efficiente(7, 0, 1, 0)    returns 21
------------------- Ending recursion -------------------
Num calls: 8
```

In [73]:
```python
# proviamo invece a lavorare in uscita dalla ricorsione
# - ciascuna chiamata ritorna la coppia *corrente, precedente*
# ovvero calcoliamo F(N) -> corrente, precedente
# - nel caso base la coppia è           1, 0
# - da corrente, precedente del mese prima (N-1) posso calcolare quelli di N
@trace
def fibonacci_efficiente(N : int ) -> tuple[int,int] :
    if N == 0:
        return 1, 0        # all'inizio ci sono 0 ed 1 coppia
    else:
        # ottengo i due valori del mese precedente (F(N-1) e F(N-2))
        corrente, precedente = fibonacci_efficiente(N-1)
        # calcolo i due valori per questo mese     (F(N)   e F(N-1))
        return corrente+precedente, corrente

print(fibonacci_efficiente(5)[0])

fibonacci_efficiente.trace(5)
```

```
8
------------------- Starting recursion -------------------
 entering       fibonacci_efficiente(5,)
|-- entering    fibonacci_efficiente(4,)
|--|-- entering fibonacci_efficiente(3,)
|--|--|-- entering      fibonacci_efficiente(2,)
|--|--|--|-- entering   fibonacci_efficiente(1,)
|--|--|--|--|-- entering        fibonacci_efficiente(0,)
|--|--|--|--|-- exiting         fibonacci_efficiente(0,)        returns (1, 0)
|--|--|--|-- exiting    fibonacci_efficiente(1,)        returns (1, 1)
|--|--|-- exiting       fibonacci_efficiente(2,)        returns (2, 1)
|--|-- exiting  fibonacci_efficiente(3,)        returns (3, 2)
|-- exiting     fibonacci_efficiente(4,)        returns (5, 3)
 exiting        fibonacci_efficiente(5,)        returns (8, 5)
------------------- Ending recursion --------------------
Num calls: 6
```

Out[73]: (8, 5)

Altro esempio: Massimo Comun Divisore di x, y interi positivi

Ovvero dobbiamo trovare quel'intero M tale che:

- $x = M * k$
- $y = M * j$

- con $k, j >= 1$ (e senza fattori comuni)

Quali sono le proprieta di x ed y?

- se **x==y** allora **k==j==1** e **M=x** (ecco un buon **caso base**!)
- altrimenti proviamo a sottrarre il minore dal maggiore
  - **z = x-y = M*(k-j)** quindi **z** e **y** hanno lo stesso **MCD**,
    e inoltre z è più piccolo di x (ecco la nostra **riduzione**!)
    - ad ogni passo si riduce la somma k+j di almeno j (il più piccolo)
    - sottraendo un numero più piccolo non si può andare nei negativi
  - a forza di sottrarre arriveremo per forza a j=k=1 ovvero al caso base (ed ecco la **convergenza**)
  - una volta trovato **M** abbiamo la soluzione di ciascun caso **più grande** (**ricomposizione**)

Ottimizzazione: invece di sottrarre y da x calcoliamone il resto -> algoritmo di **Euclide**

In [74]:

```python
# Sfruttiamo la definizione ricorsiva del problema
# per dare una implementazione ricorsiva
@trace
def GCD(x : int, y : int) -> int :
    # FIXME: controllare che siano  interi E positivi
    if x == y:
        return x
    else:
        if x>y:
            return GCD(x-y, y)
        else:
            return GCD(y-x, x)

GCD.trace(108, 64)
```

```
------------------- Starting recursion -------------------
 entering       GCD(108, 64)
|-- entering    GCD(44, 64)
|--|-- entering GCD(20, 44)
|--|--|-- entering      GCD(24, 20)
|--|--|--|-- entering   GCD(4, 20)
|--|--|--|--|-- entering        GCD(16, 4)
|--|--|--|--|--|-- entering     GCD(12, 4)
|--|--|--|--|--|--|-- entering  GCD(8, 4)
|--|--|--|--|--|--|--|-- entering       GCD(4, 4)
|--|--|--|--|--|--|--|-- exiting        GCD(4, 4)       returns 4
|--|--|--|--|--|--|-- exiting   GCD(8, 4)       returns 4
|--|--|--|--|--|-- exiting      GCD(12, 4)      returns 4
|--|--|--|--|-- exiting         GCD(16, 4)      returns 4
|--|--|--|-- exiting    GCD(4, 20)      returns 4
|--|--|-- exiting       GCD(24, 20)     returns 4
|--|-- exiting  GCD(20, 44)     returns 4
|-- exiting     GCD(44, 64)     returns 4
 exiting        GCD(108, 64)    returns 4
-------------------- Ending recursion --------------------
Num calls: 9
```

Out[74]: 4

In [75]:
```
## Non è difficile farne una versione iterativa

def GCD_iter(x : int, y : int) -> int :
    while x != y:
        if x > y:
            x -= y
        else:
            y -= x
    return x

print(GCD_iter(75,45))
```

```
15
```

Esempio: Check se una stringa/lista è palindroma (si legge uguale in senso inverso)

soluzioni iterative

- rovescio e confronto
- scandisco gli indici degli elementi agli estremi e li confronto

In [28]:
```
from typing import Sequence
## Rovescio e confronto
def palindromaP_iter1(sequenza : Sequence):
    rovesciata = sequenza[::-1]
    return rovesciata == sequenza

# leggermente inefficiente, costruisco una nuova sequenza e confronto 2N elementi
# (potrei confrontarne solo N)
palindromaP_iter1('amoRoma')
```

Out[28]:
```
True
```

In [77]:
```python
## Versione iterativa

def palindromaP_iter2(sequenza : Sequence ) -> bool :
    for i in range(len(sequenza)//2):
        print('comparing', sequenza[i], sequenza[-i-1], sequenza[i] == sequenza[-i-1]
        if sequenza[i] != sequenza[-i-1]:
            return False
    return True

palindromaP_iter2('amoRoma')
#palindromaP_iter([1, 2, 3, 4, 5, 4, 3, 2, 1])
```

```
comparing a a True
comparing m m True
comparing o o True
```

Out[77]: True

In [79]:
```python
def palindroma_iter2(sequenza):
    inizio = 0
    fine   = len(sequenza)-1
    while inizio < fine:
        print('comparing', sequenza[inizio], sequenza[fine], sequenza[inizio] == sequ
        if sequenza[inizio] != sequenza[fine]:
            return False
        inizio += 1
        fine   -= 1
    return True

palindroma_iter2([1, 2, 3, 4,  5, 4, 3, 2, 1])
```

```
comparing 1 1 True
comparing 2 2 True
comparing 3 3 True
comparing 4 4 True
```

Out[79]: True

soluzione ricorsiva

- **casi base**: ogni sequenza di lunghezza 0 o 1 è palindroma
- se è lunga 2 o più elementi il primo e l'ultimo devono essere uguali
- se non lo sono NON è palindroma (altro **caso base**)
- se lo sono deve essere palindromo anche il resto, tolto primo ed ultimo carattere

- (togliere i 2 caratteri = **riduzione**)
- a forza di togliere 2 caratteri si arrivarà sempre a 1 o 0 (**convergenza**)
- la stringa è palindroma se sono uguali primo e ultimo AND la sottostringa è palindroma (**costruzione della soluzione dalla sottosoluzione** + calcolo locale)

In [80]:
```
1
2  @trace
3  def palindromaP(sequenza : Sequence ) -> bool :
4      "predicato che verifica se una sequenza è palindroma"
5      if len(sequenza) < 2:
6          return True
7      if sequenza[0] != sequenza[-1]:
8          return False
9      else:
10         return palindromaP(sequenza[1:-1])
11
12 palindromaP.trace('amoRoma')
13
14 # un po' inefficiente perchè crea tante sottosequenze
```

```
------------------- Starting recursion -------------------
 entering       palindromaP('amoRoma',)
|-- entering    palindromaP('moRom',)
|--|-- entering palindromaP('oRo',)
|--|--|-- entering      palindromaP('R',)
|--|--|-- exiting       palindromaP('R',)       returns True
|--|-- exiting  palindromaP('oRo',)     returns True
|-- exiting     palindromaP('moRom',)   returns True
 exiting        palindromaP('amoRoma',) returns True
-------------------- Ending recursion --------------------
Num calls: 4
```

Out[80]: True

In [81]:
```python
# versione che usa gli indici inizio e fine
@trace
def _palindromaP2(sequenza : Sequence, inizio : int, fine : int ) -> bool :
    if inizio >= fine:
        return True
    if sequenza[inizio] != sequenza[fine]:
        return False
    return _palindromaP2(sequenza, inizio+1, fine-1)

def palindromaP2(sequenza):
    return palindromaP2(sequenza, 0, len(sequenza)-1)


_palindromaP2.trace('amoRoma', 0, 6)
```

```
------------------- Starting recursion -------------------
 entering       _palindromaP2('amoRoma', 0, 6)
|-- entering    _palindromaP2('amoRoma', 1, 5)
|--|-- entering _palindromaP2('amoRoma', 2, 4)
|--|--|-- entering      _palindromaP2('amoRoma', 3, 3)
|--|--|-- exiting       _palindromaP2('amoRoma', 3, 3)  returns True
|--|-- exiting  _palindromaP2('amoRoma', 2, 4)  returns True
|-- exiting     _palindromaP2('amoRoma', 1, 5)  returns True
 exiting        _palindromaP2('amoRoma', 0, 6)  returns True
-------------------- Ending recursion --------------------
Num calls: 4
```

Out[81]: True

Esploriamo un albero di directory: cerchiamo tutti i file .txt e la loro dimensione

- una directory contiene files (caso base) e sottodirectory (caso ricorsivo)
- ogni volta che esaminiamo una sottodirectory abbiamo un problema simile a quello iniziale, e più piccolo
- a forza di scendere arriveremo in una sottodirectory che contiene solo file (convergenza)
- i file trovati nelle sottodirectory vanno raccolti assieme a quelli della dir iniziale (composizione)

Per esaminare directory e files si usa la libreria **os**

In [93]:

```python
import os

@trace
def cerca_file_sizes(directory : str) -> dict[str, int] :
    "cerco tutti i file '.txt' e ne ritorno le dimensioni"
    risultato = {}
    for nome in os.listdir(directory):
        # ignoro file e dir che iniziano per '.' oppure '_'
        if nome[0] in '_.': continue
        fullname = directory + '/' + nome
        # se sono nel caso ricorsivo
        if os.path.isdir(fullname):
            trovati = cerca_file_sizes(fullname)
            risultato.update(trovati)
        # altrimenti se è un file che finisce con 'txt'
        elif nome.endswith('.txt'):
            size = os.path.getsize(fullname)
            risultato[fullname] = size
    return risultato

cerca_file_sizes.trace('../lezione11')
```

```
------------------- Starting recursion -------------------
 entering       cerca_file_sizes('../lezione11',)
|-- entering    cerca_file_sizes('../lezione11/files',)
|-- exiting     cerca_file_sizes('../lezione11/files',) returns {'../lezione11/files/test
o2.txt': 19, '../lezione11/files/holmes.txt': 594933, '../lezione11/files/alice_it.txt':
144377, '../lezione11/files/frankenstein.txt': 448684, '../lezione11/files/prince.txt': 3
05864, '../lezione11/files/testo.txt': 146, '../lezione11/files/results.txt': 272, '../le
zione11/files/alice.txt': 167517}
 exiting        cerca_file_sizes('../lezione11',)       returns {'../lezione11/files/test
o2.txt': 19, '../lezione11/files/holmes.txt': 594933, '../lezione11/files/alice_it.txt':
144377, '../lezione11/files/frankenstein.txt': 448684, '../lezione11/files/prince.txt': 3
05864, '../lezione11/files/testo.txt': 146, '../lezione11/files/results.txt': 272, '../le
zione11/files/alice.txt': 167517, '../lezione11/ita.lezione11.mkv.txt': 59078, '../lezion
e11/eng.lezione11.mkv.txt': 56956}
-------------------- Ending recursion --------------------
Num calls: 2
```

Out[93]:

```
{'../lezione11/files/testo2.txt': 19,
 '../lezione11/files/holmes.txt': 594933,
 '../lezione11/files/alice_it.txt': 144377,
 '../lezione11/files/frankenstein.txt': 448684,
 '../lezione11/files/prince.txt': 305864,
 '../lezione11/files/testo.txt': 146,
 '../lezione11/files/results.txt': 272,
```

```
'../lezione11/files/alice.txt': 167517,
'../lezione11/ita.lezione11.mkv.txt': 59078,
'../lezione11/eng.lezione11.mkv.txt': 56956}
```

In [87]:
```python
## soluzione ricorsiva che raccoglie i file
## in un dizionario fornito come argomento

def cerca_file_sizes2(directory : str, dizionario : dict[str, int]) -> None:
    "cerco tutti i file '.txt' e ne ritorno le dimensioni"
    for nome in os.listdir(directory):
        # ignoro file e dir che iniziano per '.' oppure '_'
        if nome[0] in '_.': continue
        fullname = directory + '/' + nome
        # se sono nel caso ricorsivo
        if os.path.isdir(fullname):
            cerca_file_sizes2(fullname, dizionario)
        # altrimenti se è un file che finisce con 'txt'
        elif nome.endswith('.txt'):
            size = os.path.getsize(fullname)
            dizionario[fullname] = size

D = {}
cerca_file_sizes2('..', D)
D
```

Out[87]:
```
{'../lezione10/files/testo2.txt': 19,
 '../lezione10/files/holmes.txt': 594933,
 '../lezione10/files/alice_it.txt': 144377,
 '../lezione10/files/frankenstein.txt': 448684,
 '../lezione10/files/prince.txt': 305864,
 '../lezione10/files/testo.txt': 146,
 '../lezione10/files/results.txt': 272,
 '../lezione10/files/alice.txt': 167517,
 '../lezione10/dati.txt': 53,
 '../lezione10/pippo.txt': 71,
 '../lezione10/topolino.txt': 71,
 '../lezione10/en.lezione10.mkv.txt': 80332,
 '../lezione10/it.lezione10.mkv.txt': 85229,
 '../lezione13/eng.lezione.13.txt': 57126,
 '../lezione13/ita.lezione.13.txt': 61794,
 '../lezione08/en.lezione08.ogg.txt': 64027,
 '../lezione08/5min.ogg.txt': 4241,
 '../lezione08/lezione08.ogg.txt': 69085,
 '../lezione14/ita.lezione.14.txt': 60828,
 '../lezione14/eng.lezione.14.txt': 57012,
 '../lezione15/eng.lezione.15.txt': 64808,
 '../lezione15/ita.lezione.15.txt': 70661,
 '../lezione03/en.lezione03.ogg.txt': 27230,
 '../lezione03/lezione03.mp4.txt': 79307,
 '../lezione02/en.lezione02.ogg.txt': 43464,
```

'../lezione02/lezione02-2022-10-03_10-13-30.mkv.txt': 119406,
'../lezione05/en.lezione05.ogg.txt': 72679,
'../lezione05/lezione05.mkv.txt': 63000,
'../lezione01/en.lezione01.ogg.txt': 16041,
'../lezione01/lezione01.mp4.txt': 17238,
'../lezione11/files/testo2.txt': 19,
'../lezione11/files/holmes.txt': 594933,
'../lezione11/files/alice_it.txt': 144377,
'../lezione11/files/frankenstein.txt': 448684,
'../lezione11/files/prince.txt': 305864,
'../lezione11/files/testo.txt': 146,
'../lezione11/files/results.txt': 272,
'../lezione11/files/alice.txt': 167517,
'../lezione11/ita.lezione11.mkv.txt': 59078,
'../lezione11/eng.lezione11.mkv.txt': 56956,
'../lezione07/lezione07.ogg.txt': 82839,
'../lezione07/en.lezione07.ogg.txt': 75637,
'../lezione06/lezione06.mkv.txt': 40737,
'../lezione06/en.lezione06.ogg.txt': 2680}