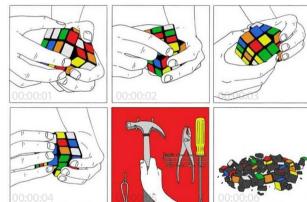




Algoritmi in Python 3



Introduzione alla programmazione

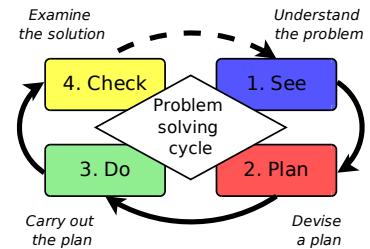
Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Problem solving

- George Polya (1945). *"How to solve it"*
 - Soluzione di problemi matematici: processo raramente lineare
- **(1) See.** Capire il problema
 - Quali sono i dati, quali le incognite?
 - Quali sono le condizioni? Sono soddisfacibili, ridondanti, contraddittorie?
 - Figure, notazione

"Make things as simple as possible, but not simpler. (A. Einstein)"

"For every complex problem there is an answer that is clear, simple, and wrong. (H.L. Mencken)"



www.ce.unipr.it/people/tomamic

2/43

Dal problema alla soluzione, e ritorno

- **(2) Plan.** Elaborare un piano
 - Mettere in relazione dati e incognite
 - Riduzione, analogia, divide et impera, composizione, astrazione...
 - *Computational thinking*
 - Cominciare a risolvere un problema *più semplice* (vincoli rilassati)
- **(3) Do.** Eseguire il piano
 - Controllare ogni passo. È corretto?
- **(4) Check.** Controllare la soluzione
 - Corretta? Ottenibile in altro modo?
 - Risultato, o metodo, utilizzabile per altri problemi?

"If you can't solve a problem... then there is an easier problem you can solve: find it. (G. Polya)"

Elementi di un algoritmo

- **Dati:** iniziali (istanza problema), intermedi, finali (soluzione)
- **Passi** elementari: azioni atomiche non scomponibili in azioni più semplici
- **Processo**, o anche esecuzione: sequenza ordinata di passi
- **Proprietà:** finitezza, non ambiguità, realizzabilità, efficienza...

"Computer science is no more about computers than astronomy is about telescopes. (E. Dijkstra...)"



Una ricetta di cucina è un esempio di algoritmo

www.ce.unipr.it/people/tomamic

3/43

www.ce.unipr.it/people/tomamic

4/43

Ricerca in uno schedario

- Es. in biblioteca: cercare la scheda di un certo libro
 - (1) **Finchè** restano delle schede da esaminare: si prende la prima scheda non ancora controllata
 - (2) **Se** autore e titolo sono quelli cercati: scheda trovata, ricerca conclusa!
 - (3) (Altrimenti...)
Si ripete il controllo (1), passando alla scheda successiva
 - (4) Esaurite le schede, il libro non è nella biblioteca!



Cercare più velocemente

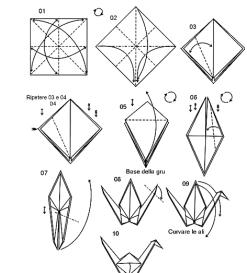
- Su schedario **ordinato** si può fare più in fretta
 - (1) **Finchè** restano delle schede da esaminare: si prende tra loro la scheda centrale
 - (2) **Se** autore e titolo sono quelli cercati: scheda trovata, ricerca conclusa!
 - (3) **Altrimenti, se** autore e titolo vengono dopo quelli cercati: si scartano subito tutte le schede successive
 - (4) **Altrimenti, infine:** si scartano le schede precedenti
 - (5) **Si ripete** la ricerca sull'insieme dimezzato (1)
 - (6) Esaurite le schede, il libro non è nella biblioteca!

Complessità e calcolabilità

- Complessità:** classificare algoritmi (e problemi)
 - Trattabili:** costo accettabile, "polinomiale"
 - Non trattabili:** costo "esponenziale"
- Calcolabilità:** distinguere i problemi **non risolvibili**
 - Es. Valore di verità di P: *Questa frase è falsa*
 - Incompletezza Gödel; indecidibilità *terminazione*
 - ∀ formalizzazione della matematica che assiomatizza \mathbb{N}
→ ∃ proposizione né dimostrabile né confutabile

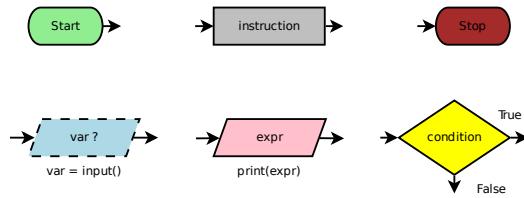
Diagramma di flusso

- Flow-chart:** Rappresentazione grafica di algoritmi
 - Più efficace e meno ambigua di una descrizione a parole
- Due tipi di entità:
 - Nodi
 - Archi
- È un **grafo orientato**
 - Passi di un algoritmo e loro sequenza

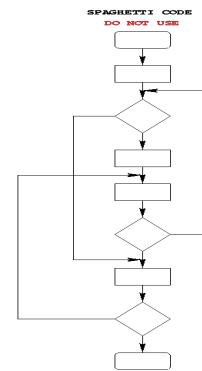


Anche un origami è un esempio di algoritmo

Tipi di nodi



- Istruzioni di I/O: es. leggere dati da tastiera o mostrarli a schermo
 - Operazioni aritmetico-logiche
 - Controllo del flusso di esecuzione



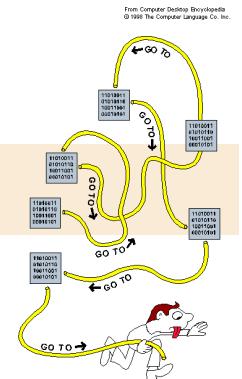
9/43

Programmazione strutturata

- Strutture di controllo:
 - Sequenza
 - Selezione
 - Iterazione

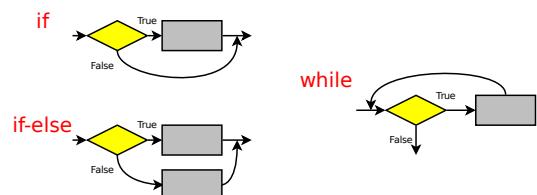
“Qualunque algoritmo può essere implementato utilizzando queste tre sole strutture (Teorema di Böhm-Jacopini, 1966)”

“Goto statement considered harmful (Dijkstra, 1968)



10/43

Strutture di controllo



- Esempi quotidiani di `if` e `while`:
 - *"Se non c'è il lievito, usare due cucchiaini di bicarbonato"*
 - *"Battere gli albumi finché non montano"*



[Blockly](#) > [Demos](#) > Maze

A Scratch script for a robot to solve a maze. The script uses a combination of `repeat`, `while`, and `if` loops along with `turn left` and `turn right` commands to navigate through the maze.

```
when green flag clicked
  [repeat [wall-ahead?]
    [do [move forward] until [wall-ahead?]
    if [then [turn left] [and [repeat [turn right]
      [do [ ] until [ ]]] until [ ]]] end
  end]
```

<https://blockly-games.appspot.com/maze>



11/43

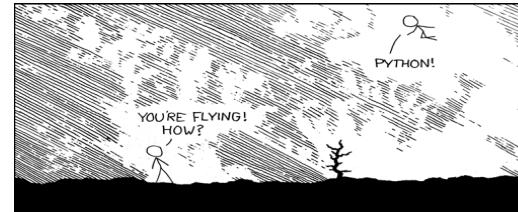


12/43

Python!



Linguaggio Python



Applicazioni in Python

Why?

Who uses Python ?:

- Google
- NASA
- Yahoo
- Youtube
- Linux (RedHat, Ubuntu, ...)
- Lots of researchers
- EVE online (Thousands of online players)
- MIT (Programming Intro. Course)
- etc...

YouTube

Tipi di dati

- Un **tipo di dato** specifica un insieme di *valori* e le *operazioni* ammesse
 - int, float, bool, str
 - Operazioni aritmetiche e logiche, confronti
- Una **variabile** serve per conservare un risultato

```
3 + 4 # 7 - The result has to be handled...
4 == 5 # False - Do not confuse with assignment!
2 < 3 or not True # True
```

PYTHON

```
a = 5 # Assignment
b = 2
a = a + b
```

PYTHON

- Web, analisi di dati, scripting, insegnamento, giochi, hardware, multipiattaforma...

Valori numerici e booleani

- **int** o **float**, per numeri interi o reali
 - Operazioni di base: +, -, *, /
 - Divisione intera, resto, potenza: //, %, **
 - Assegnamento: =, +=, -=, ...
 - Confronti: <, <=, >, >=, ==, !=
- **bool**, per valori booleani: **True**, **False**
 - Operazioni consentite: **and**, **or**, **not**
 - I confronti danno un risultato booleano

```
-2 // 3    # -1 (floored integer division)
-2 % 3    # 1 (reminder is always positive)
2 ** 1000 # no limits (but memory)
```

PYTHON

Stringhe di testo

- **str** per sequenze di caratteri **Unicode**
- Primi 128 codici **Unicode == ASCII**
 - A capo: '\n' (10, o 13-10... ma conversione automatica)
 - Tabulazione: '\t' (9)
- Confronto tra stringhe, in ordine **lessicografico**
 - <, <=, >, >=, ==, !=

```
str1 = "Monty Python's "
str2 = 'Flying Circus'
result = str1 + str2

'first' < 'second' # True (but... 'Second' < 'first')
chr(90)           # 'Z'
ord('Z')          # 90
```

PYTHON

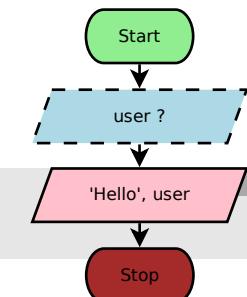
Tabella ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	(NULL)	22	1A	[SPACE]	64	40	@	96	60	'
1	1	(START OF HEADING)	33	21	!	65	41	A	97	61	a
2	2	(START OF TEXT)	34	22	"	66	42	B	98	62	b
3	3	(END OF TEXT)	35	23	#	67	43	C	99	63	c
4	4	(END OF TRANSMISSION)	36	24	\$	68	44	D	100	64	d
5	5	(ENQ)	37	25	%	69	45	E	101	65	e
6	6	(ACKNOWLEDGE)	38	26	&	70	46	F	102	66	f
7	7	(BELL)	39	27	'	71	47	G	103	67	g
8	8	(BACKSPACE)	40	28	(72	48	H	104	68	h
9	9	(HORIZONTAL TAB)	41	29)	73	49	I	105	69	i
10	A	(VERTICAL TAB)	42	2A	*	74	4A	J	106	6A	j
11	B	(VERTICAL TAB)	43	2B	+	75	4B	K	107	6B	k
12	C	(FORM FEED)	44	2C	,	76	4C	L	108	6C	l
13	D	(CARRIAGE RETURN)	45	2D	.	77	4D	M	109	6D	m
14	E	(SHIFT OUT)	46	2E	.	78	4E	N	110	6E	n
15	F	(SHIFT IN)	47	2F	/	79	4F	O	111	6F	o
16	10	(DOWN SHIFT ESCAPE)	48	30	0	80	50	P	112	70	p
17	11	(DEVICE CONTROL J)	49	31	1	81	51	Q	113	71	q
18	12	(DEVICE CONTROL J)	50	32	2	82	52	R	114	72	r
19	13	(DEVICE CONTROL J)	51	33	3	83	53	S	115	73	s
20	14	(DEVICE CONTROL J)	52	34	4	84	54	T	116	74	t
21	15	(NEGATIVE ACKNOWLEDGE)	53	35	5	85	55	U	117	75	u
22	16	(SYNCHRONOUS IDLE)	54	36	6	86	56	V	118	76	v
23	17	(ENQ OF TRANS. BLOCK)	55	37	7	87	57	W	119	77	w
24	18	(CANCEL)	56	38	8	88	58	X	120	78	x
25	19	(END OF MEDIUM)	57	39	9	89	59	Y	121	79	y
26	1A	(SUBSTITUTE)	58	3A	:	90	5A	Z	122	7A	z
27	1B	(ESCAPE)	59	3B	;	91	5B	[123	7B	{
28	1C	(TAB SEPARATOR)	60	3C	<	92	5C	\	124	7C	l
29	1D	(GROUP SEPARATOR)	61	3D	=	93	5D	_	125	7D]
30	1E	(RECORD SEPARATOR)	62	3E	>	94	5E	^	126	7E	~
31	1F	(UNIT SEPARATOR)	63	3F	?	95	5F	-	127	7F	[DEL]

Leggere e scrivere

- **input** legge una riga di testo, inserita dall'utente, in una **variabile**
 - Prima mostra un messaggio
- **print** scrive una serie di valori su una riga
 - Tra i valori (parametri) viene inserito uno spazio

```
user = input("What's your name? ")
print("Hello, ", user)
```

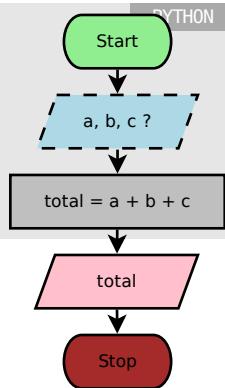


Somma di tre numeri

```
# File sum3.py - to run: python3 sum3.py
a = int(input("Insert 1st val: "))
b = int(input("Insert 2nd val: "))
c = int(input("Insert 3rd val: "))

total = a + b + c

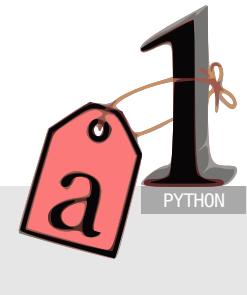
print("The sum is", total)
```



Variabile

- **Nome** (etichetta) associato ad un certo **valore** (oggetto)
 - Oggetto assegnato a più variabili: non viene copiato, ma riceve più etichette
 - Il **tipo** dipende dal valore attualmente assegnato
 - Una var. non dev'essere *dichiarata*, ma dev'essere *inizializzata*

```
x = 100
DELTA = 5      # Constants: UPPER_CASE
x = x + DELTA # Variables: lower_case
next_position = x # Use explicative names!
```



Strutture di controllo

Selezione: if

- **Indentazione** del corpo di **if** o **else**
 - Richiesta per sintassi, non opzionale!

"Readability counts (The Zen of Python)"

```
# File tooyoung.py - to run: python3 tooyoung.py
age = int(input("Age? "))

if age < 14:
    print("You're too young for driving a scooter...")
    print("But not for learning Python!")
```

PYTHON

- Clausola **else**: opzionale
 - Eseguita se la condizione non è verificata

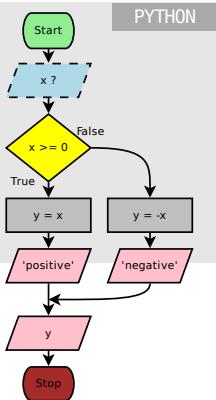
Valore assoluto

```
x = int(input("insert a value: "))

if x >= 0:
    y = x
    print(x, "is positive")
else:
    y = -x
    print(x, "is negative")

print("abs =", y)
```

- Corpo di **if o else**: qualsiasi istruzione
- Anche altri blocchi **if o while** annidati!



Calcolo dell'età

```
birth_year = int(input("Birth year? "))
birth_month = int(input("Birth month? "))
birth_day = int(input("Birth day? "))
current_year = int(input("Current year? "))
current_month = int(input("Current month? "))
current_day = int(input("Current day? "))

if (current_month > birth_month
    or (current_month == birth_month and current_day >= birth_day)):
    age = current_year - birth_year
else:
    age = current_year - birth_year - 1

print("Your age is", age)
```

Espressione booleana composta con **and e or**

www.ce.unipr.it/people/tomamic

PYTHON

26/43

www.ce.unipr.it/people/tomamic

25/43

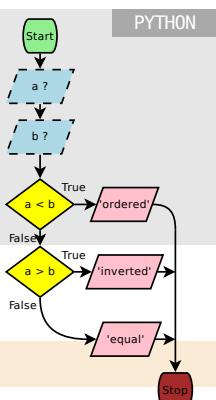
Confronto tra parole

```
a = input("First word? ")
b = input("Second word? ")

if a < b:
    print("The words are ordered")
elif a > b:
    print("The words are inverted")
else:
    print("The words are equal")

• elif: clausola else che contiene un altro if
• No switch, no do-while
```

"There should be one -- and preferably only one -- obvious way to do it (ZoP)"

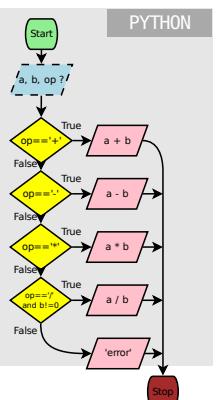


Operazioni aritmetiche

```
a = float(input())
b = float(input())
op = input()

if op == '+':
    print(a + b)
elif op == '-':
    print(a - b)
elif op == '*':
    print(a * b)
elif op == '/' and b != 0:
    print(a / b)
else:
    print("Operation not allowed")
```

www.ce.unipr.it/people/tomamic



www.ce.unipr.it/people/tomamic

27/43

28/43

Iterazione: while

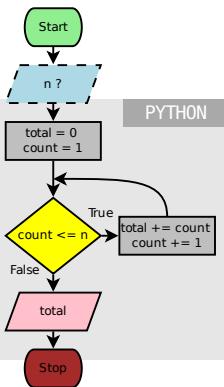
- Condizione booleana di *permanenza* nel ciclo
- Controllo *preliminare*, possibile che il corpo non sia mai eseguito

```
# Sum of the numbers from 1 to n
total = 0
count = 1
n = int(input("n? "))

while count <= n:
    total = total + count
    count = count + 1

print("The sum is", total)
```

<http://it.wikipedia.org/wiki/Gauss>



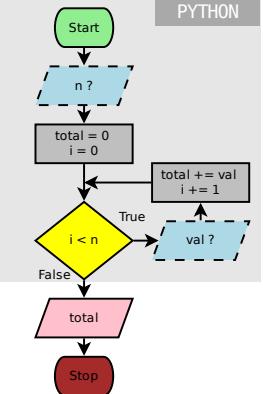
Somma di N valori dell'utente

```
n = int(input("How many values? "))
total = 0
i = 0

while i < n:
    val = int(input("Val? "))

    total += val # total = total + val
    i += 1 # i = i + 1

print("The sum is", total)
```



www.ce.unipr.it/people/tomamic

29/43

www.ce.unipr.it/people/tomamic

30/43

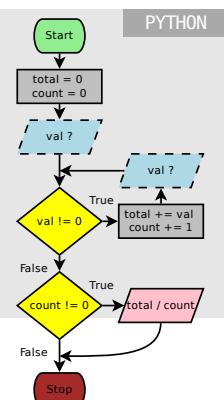
Sequenza terminata da zero

```
total = 0
count = 0

val = int(input("Val? (0 to finish) "))

while val != 0:
    total += val
    count += 1
    val = int(input("Val? (0 to finish) "))

if count != 0:
    print("The average is", total / count)
```

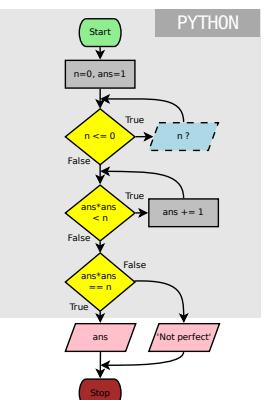


Quadrato perfetto

```
n = 0
while n <= 0:
    n = int(input("Positive val? "))

ans = 1
while ans * ans < n:
    ans += 1

if ans * ans == n:
    print("Square root:", ans)
else:
    print("Not a perfect square")
```



www.ce.unipr.it/people/tomamic

31/43

www.ce.unipr.it/people/tomamic

32/43



Esercizi

Cerchio

- Chiedere all'utente il valore del raggio r di un cerchio
 - Mostrare il valore dell'area e della circonferenza
 - Se r è negativo, però, mostrare un messaggio d'errore

```
from math import pi  
# `pi` is a constant defined in the `math` module
```

	3.14159265358979323846264338327
	0288479165938375150529744945923
	07834606084589938621978463907724117067
9823	8132
823	65647
823	65647
17	23598
	4081
	2848
	137
	4502
	8410
	2701
	9385
	2110
	5564
	46229
	49594
	9302
	81364
	44688
	10765
	65593
	34461
	20903
	4423
	78878
	31652
201901	71
	454460
	66
	453333
	07260454326117
2333936	07260429431427
3724587	00606361558



 www.ce.unipr.it/people/tomamic

33/43

 www.ce.unipr.it/people/tomamic

34/43

Minore e maggiore

- Chiedere all'utente tre numeri interi: **a, b, c**
 - Determinare qual è il minore dei tre
 - Determinare qual è il maggiore dei tre

*Controllare prima di tutto se a è minore degli altri due
Altrimenti controllare se b è minore di c
Altrimenti ...*



Cubi in ciclo

- In un ciclo, ripetere le seguenti operazioni
 - Chiedere all'utente un numero
 - Mostrare il suo valore al cubo
 - Il valore 0 indica il termine della sequenza



 www.ce.unipr.it/people/tomamic

35/43

 www.ce.unipr.it/people/tomamic

36/43

Numero segreto

- Generare un intero "segreto" a caso tra 1 e 90
- Chiedere ripetutamente all'utente di immettere un numero, finché non indovina quello generato
- Ad ogni tentativo, dire se il numero immesso è maggiore o minore del numero segreto

```
from random import randint
secret = randint(1, 90)
# `randint` is a function defined in the `random` module
```



Lunghezza righe (*)

- Leggere una sequenza di righe di testo, in un ciclo
- La sequenza termina con una riga vuota
- Visualizzare la lunghezza media delle righe

Lunghezza di una variabile line di tipo stringa: len(line)



Interesse composto (*)

- Dati dall'utente: capitale iniziale, tasso d'interesse, durata investimento
- Calcolare il capitale dopo ogni anno
- Es. 100€ al 4.5%:

```
Anno 0: 100.00€
Anno 1: 104.50€
Anno 2: 109.20€
Anno 3: 114.12€ ...
```



Conteggio cifre (*)

- Leggere un numero intero positivo
- Determinare di quante cifre è composto

Quante volte riusciamo a dividerlo per 10, prima che si annulli?



Calendario a muro (*)

- L'utente inserisce:
 - La lunghezza del mese (28-31)
 - Il primo giorno del mese (lunedì=0 ... domenica=6)
- Visualizzare il calendario del mese
 - In prima colonna la data
 - In seconda colonna il giorno della settimana (0-6)

Procedere una riga alla volta (ltr →, ttb ↓)
Sfruttare l'operatore % (resto della divisione)



Resistenze, ciclo (*)

- Leggere, attraverso un ciclo, una sequenza di valori di resistenze elettriche
- La sequenza termina quando l'utente immette il valore 0
- Alla fine, visualizzare la resistenza equivalente, sia nel caso di resistenze disposte in serie, che in parallelo



Formule utili:

$$R_s = \sum R_i$$

$$1/R_p = \sum (1/R_i)$$

<Domande?>

Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR



Introduzione alla
programmazione

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR



Sequenze di dati

Lista

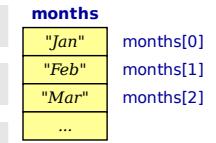
- Sequenza di elementi, *di solito* dello stesso **tipo**
 - L'intera lista può essere assegnata ad una variabile, così diamo un **nome** alla lista
- I singoli elementi sono **numerati**
 - Gli indici partono da 0!



```
to_buy = ['spam', 'eggs', 'beans']
```

```
rainfall_data = [13, 24, 18, 15]
```

```
months = ['Jan', 'Feb', 'Mar',
          'Apr', 'May', 'Jun',
          'Jul', 'Aug', 'Sep',
          'Oct', 'Nov', 'Dec']
```



Accesso agli elementi

- Attenzione ad usare indici validi!**
 - Lunghezza* attuale di una lista x: `len(x)`
 - Elementi **numerati** da 0 a `len(x) - 1`
 - Indici **negativi** contano dalla fine



```
n = len(months)           # 12
months[3]                  # 'Apr'
months[-2]                 # 'Nov', same as n - 2
```

```
to_buy = ['spam', 'eggs', 'beans']
```

```
to_buy[1] = 'ketchup'      # replace an element
```

PYTHON

Appartenenza, inserimento, rimozione

```
to_buy = ['spam', 'eggs', 'beans']
'eggs' in to_buy      # True, to_buy contains 'eggs'
```

PYTHON

```
to_buy.append('bacon')    # add an element to the end
to_buy.pop()              # remove (and return) last element

to_buy.insert(1, 'bacon')  # other elements shift
removed = to_buy.pop(1)   # remove (and return) element at index

to_buy.remove('eggs')     # remove an element by value
```

PYTHON

Slice: porzioni di lista

```
spring = months[2:5]        # ['Mar', 'Apr', 'May']
quart1 = months[:3]         # ['Jan', 'Feb', 'Mar']
quart4 = months[-3:]        # ['Oct', 'Nov', 'Dec']
whole_year = months[:]      # Copy the whole list
```

PYTHON

```
list1 = ['spam', 'eggs', 'beans']
list2 = ['sausage', 'mushrooms']
to_buy = list1 + list2      # List concatenation
```

PYTHON

```
so_boring = [1, 2] * 3      # List repetition:
                            # [1, 2, 1, 2, 1, 2]
results_by_month = [0] * 12
```

PYTHON

Uguaglianza e identità

```
a = [3, 4, 5]
b = a[:]      # b = [3, 4, 5] -- a new list!
b == a        # True, they contain the same values
b is a        # False, they are two objects in memory
              # (try and modify one of them...)
c = a
c is a        # True, same object in memory
              # (try and modify one of them...)
```

PYTHON

Stringhe e liste

- **Stringa**: sequenza *immutable* di caratteri
- **join** e **split**: da lista a stringa e viceversa

```
txt = "Monty Python's Flying Circus"
txt[0]  # 'M'
txt[1]  # 'o'
txt[-1] # 's'
txt[6:12] # 'Python'
txt[-6:] # 'Circus'

days = ['tue', 'thu', 'sat']
txt = '|'.join(days) # 'tue|thu|sat'

days = 'mon|wed|fri'.split('|')
# ['mon', 'wed', 'fri']
```

PYTHON

Cicli su liste: for

```
shopping_list = ['spam', 'eggs', 'bacon', 'ketchup']

print('Your shopping list contains:')

for product in shopping_list:
    print(product)
```

- Ad ogni iterazione, a **product** è assegnato un diverso elemento della lista **shopping_list**
- Si può usare un ciclo **for** su qualsiasi sequenza, anche su una **stringa**

PYTHON

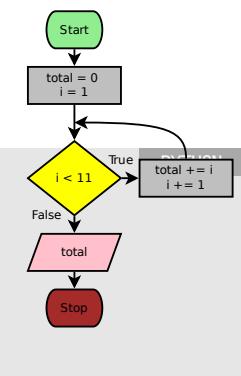
Intervalli di valori: range

- **range**: intervallo di valori aperto a destra
 - Estremo inferiore *incluso*
 - Estremo superiore *escluso*
 - Iterabile con un ciclo **for**

```
# Add up numbers from 1 to 10

total = 0
for i in range(1, 11):
    total += i

# total = 0; i = 1
# while i < 11:
#     total += i; i += 1
```



Cicli e annidamento

```
MAX = 12
y = int(input("Insert a value: "))
for x in range(1, MAX + 1):
    print(x * y, end=' ')
    # inserts a space, instead of a newline
```

```
MAX = 12
for y in range(1, MAX + 1):
    for x in range(1, MAX + 1):
        val = x * y
        print(val, end=' ')
print()
```

Per fare in modo che ciascun valore richieda almeno 3 caratteri:
val = '{:3}'.format(x * y)

TAVOLA PITAGORICA												
1	2	3	4	5	6	7	8	9	10	11	12	
2	4	6	8	10	12	14	16	18	20	22	24	
3	6	9	12	15	18	21	24	27	30	33	36	
4	8	12	16	20	24	28	32	36	40	44	48	
5	10	15	20	25	30	35	40	45	50	55	60	
6	12	18	24	30	36	42	48	54	60	66	72	
7	14	21	28	35	42	49	56	63	70	77	84	
8	16	24	32	40	48	56	64	72	80	88	96	
9	18	27	36	45	54	63	72	81	90	99	108	
10	20	30	40	50	60	70	80	90	100	110	120	
11	22	33	44	55	66	77	88	99	110	121	132	
12	24	36	48	60	72	84	96	108	120	132	144	

List comprehension

- Modo conciso per creare una lista
- Ogni elemento: risultato di una operazione su un membro di altro iterabile
- Condizione sugli elementi, opzionale

```
squares = [x ** 2 for x in range(12)]
# squares = []
# for x in range(12):
#     squares.append(x ** 2)
```

```
even_nums = [str(x) for x in range(12) if (x % 2) == 0]
```

PYTHON

PYTHON

Tupla

- Sequenza **immutable** di valori, anche di *tipo diverso*

```
# Tuple packing
pt = 5, 6, 'red'
pt[0] # 5
pt[1] # 6
pt[2] # 'red'

# multiple assignments, from a tuple
x, y, colour = pt # sequence unpacking
a, b = 3, 4
a, b = b, a
```

PYTHON

Rettangoli e cerchi con Pygame

```
import pygame
pygame.init() # Prepare pygame
screen = pygame.display.set_mode((640, 480)) # (w, h)
screen.fill((255, 255, 255)) # BG (Red, Green Blue)

# Yellow rectangle, left=50, top=75, w=90, h=50
pygame.draw.rect(screen, (255, 255, 0), (50, 75, 90, 50))

# Blue circle, center=(300, 50), radius=20
pygame.draw.circle(screen, (0, 0, 255), (300, 50), 20)

pygame.display.flip() # Update the screen
while pygame.event.wait().type != pygame.QUIT:
    pass
pygame.quit()
```



The 2D Screen Coordinates: The origin is located at the top-left corner, with x-axis pointing left and y-axis pointing down.

→ pygame.info



Dizionari, matrici, file

Dizionario

- A volte chiamato *mappa* o *array associativo*
- Insieme non ordinato di coppie chiave / valore
 - Le chiavi sono *uniche*: come nelle liste fanno da *indice* per accedere al valore corrispondente
 - Ma possono essere **int** o **str** (o altro tipo immutabile)



PYTHON

```
tel = {'john': 4098, 'terry': 4139}
tel['graham'] = 4127
tel # {'graham': 4127, 'terry': 4139, 'john': 4098}

list(tel.keys()) # ['graham', 'terry', 'john']
```

Liste multidimensionali

```
a = [['A', 'B', 'C', 'D'],
      ['E', 'F', 'G', 'H'],
      ['I', 'L', 'M', 'N']]      # 2D

b = ['A', 'B', 'C', 'D',
      'E', 'F', 'G', 'H',
      'I', 'L', 'M', 'N']      # 1D

i = y * cols + x          # 2D -> 1D

y = i // cols
x = i % cols              # 1D -> 2D
```

PYTHON

Somma colonne: matrice

```
matrix = [[2, 4, 3, 8],
          [9, 3, 2, 7],
          [5, 6, 9, 1]]
rows = len(matrix)
cols = len(matrix[0])

for x in range(cols):
    total = 0
    for y in range(rows):
        val = matrix[y][x]
        total += val
    print('Col #', x, 'sums to', total)
```

PYTHON

Lista come pseudo-matrice

```
matrix = [2, 4, 3, 8,
          9, 3, 2, 7,
          5, 6, 9, 1]
rows = 3 # Cannot be guessed from matrix alone
cols = len(matrix) // rows

for x in range(cols):
    total = 0
    for y in range(rows):
        val = matrix[y * cols + x] # 2D -> 1D
        total += val
    print('Col #', x, 'sums to', total)
```

PYTHON

Matrici di dimensioni note

```
blank_matrix = [
    ' ' * cols for y in range(rows) ]
# blank_matrix = []
# for y in range(rows):
#     blank_matrix.append([' '] * cols)
```

PYTHON

Nota. Se si usasse la repetition per replicare le righe, si otterrebbero più riferimenti alla stessa unica riga.

Flussi di dati

- **Stream:** astrazione per flussi di informazione
 - Lettura o scrittura di informazioni su *qualsiasi* dispositivo I/O (*file, ma non solo*)
- **File di testo**
 - Varie codifiche (*UTF-8* o altro)
 - Conversioni automatiche, es. "\n" → "\r\n"
- **File binari**
 - I/O preciso byte a byte, senza nessuna conversione
 - Qualsiasi file... anche di testo!



Scrittura su file

- Funzione **open** per accedere ad un file (di testo)
 - Modalità scrittura o lettura: 'w', o 'r'
- Scrittura su file: funzione **print**, o metodo **write**
- Blocco **with**: chiude il file al termine delle operazioni (anche in caso di errore)

```
with open('some_file.txt', 'w') as f:
    f.write('First line\n') # explicit newline
    print('Second line', file=f)
    # continue writing here...
```

PYTHON

Lettura da file

```
with open('some_file.txt', 'r') as f1:  
    first_line = f1.readline()  
    second_line = f1.readline()  
    # both strings contain '\n' at the end  
    # at end of file, an empty string is read  
  
with open('other_file.txt', 'r') as f2:  
    whole_text = f2.read()  
    # do stg with whole_text  
  
with open('last_file.txt', 'r') as f3:  
    for line in f3:  
        # line contains '\n' at the end  
        # strip() removes whitespaces at both ends  
        print(line.strip(), ':', len(line))
```

PYTHON

I/O su stringhe e console

- Stringhe come stream: `io.StringIO`
- Console come stream: `sys.stdin, sys.stdout, sys.stderr`

```
import io, sys
```

PYTHON

```
with io.StringIO() as output:  
    output.write('First line.\n')  
    print('Second line.', file=output)  
    # Retrieve stream contents, i.e. 'First line.\nSecond line.\n'  
    contents = output.getvalue()  
    sys.stdout.write(contents)
```

```
for line in sys.stdin: # CTRL-D (Lin) or CTRL-Z (Win) to end the input  
    print(len(line)) # notice '\n' at the end
```

PYTHON

Errori da file

- Eccezioni:** per gestire separatamente i casi inattesi
 - Errore all'interno di `try`: esecuzione interrotta subito
 - Eseguito il blocco `except` che gestisce il tipo di errore verificatosi (possibile avere diversi blocchi `except`)
 - Il blocco `with` assicura la chiusura del file



```
try:  
    with open('other_file.txt', 'r') as f:  
        whole_text = f.read()  
        # do stg with whole_text  
except IOError as err:  
    print('Oh, my!')
```

PYTHON



Esercizi

Conteggio spazi

- Chiedere una riga di testo all'utente
- Contare il numero di spazi presenti
- Se si incontra un carattere di tabulazione, contarlo come 4 spazi

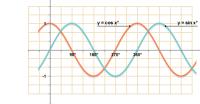
Usare un ciclo for sulla stringa (sequenza di caratteri)



Array, precalcolo

- Riempire una lista con i valori di $\sin(x)$
 - 360 elementi, indice x tra 0 e 359
- Chiedere ripetutamente un angolo all'utente, visualizzare il corrispondente valore precalcolato

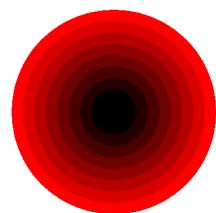
`sin, pi in modulo math`
`sin opera su radianti: rad = x * pi / 180`



Estensione opzionale: salvare la lista di valori in un file e ricaricarla all'avvio, se già disponibile

Cerchi concentrici

- Chiedere all'utente il numero di cerchi da disegnare
- Disegnare i cerchi con raggio decrescente, ma tutti con lo stesso centro
- Far variare il colore dei cerchi
 - Dal rosso del livello più esterno
 - Fino al nero del livello più interno



Cominciare a disegnare un grosso cerchio rosso

Poi, inserire l'operazione di disegno un cerchio, togliendo ad ogni passo 10 (p.es. al raggio e al livello di rosso)

Infine, determinare automaticamente, prima del ciclo, le variazioni migliori per raggio e colore

Memory

- Allocare una lista, di dimensione `rowsxcols`
 - L'utente sceglie `rows` e `cols`, però celle in numero pari
- Inserire in ordine le prime lettere dell'alfabeto, ciascuna ripetuta due volte
- Mescolare le celle
 - Per ciascuna cella, scegliere una posizione a caso e scambiare il contenuto delle celle
- Mostrare all'utente la lista risultante, andando a capo per ogni riga



Usare una lista semplice, ma nella visualizzazione introdurre dei ritorni a capo

Cella a inizio riga: il suo indice i è multiplo di `cols`, ossia $i \% cols == 0$
Cella a fine riga: $i \% cols == cols - 1$

Per cominciare, inserire nella lista valori numerici crescenti, anziché lettere

Scitala spartana

- Leggere un intero file di testo
- Inserire in una matrice i primi $W \times H$ caratteri
 - W colonne \times H righe, valori prefissati
 - Riempire una riga della matrice dopo l'altra
 - Da destra a sinistra, una riga alla volta (\rightarrow, \downarrow)
- Scrivere il contenuto della matrice su console
 - Scrivere una colonna della matrice dopo l'altra
 - Prima riga su console = prima colonna della matrice...
 - Dall'alto verso il basso, una colonna alla volta (\downarrow, \rightarrow)

Usare una lista di liste (con dimensioni predefinite)



<Domande?>

Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR

Funzioni



Introduzione alla programmazione

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

PYTHON

```
def add(a, b):
    s = a + b
    return s

def append_squares(data, num):
    # procedure: process data, no direct result
    for val in range(num):
        data.append(val ** 2)
```

Chiamata di funzioni

- **def** definisce una funzione, ma non la esegue!
- Bisogna **chiamarla**
- Ogni funzione, quando eseguita, crea un nuovo **spazio di nomi**
 - Parametri e variabili hanno **ambito locale**
 - Non visibili nel resto del programma
 - Anche nomi uguali, definiti in ambiti diversi, restano distinti

```
x = int(input('1st val? '))
y = int(input('2nd val? '))
z = add(x, y)
print(z)

values = []
append_squares(values, 4)
print(values)
```

PYTHON

Parametri di funzioni

- **Parametri formali:** nomi usati nella **definizione**
- **Parametri effettivi:** oggetti passati alla funz.
- Parametri passati "**per oggetto**"
 - Variabili all'esterno: non vengono modificate
 - Liste e oggetti passati ad una funz.: modificherebbero i **permanent**i
- Si possono restituire più valori, come **tupla**
 - **return 7, 5, 'black'**

```
def inc(a):
    a += 1
    print(a) # just for debug
x = 10
inc(x)
print(x) # just for debug
```

PYTHON

Documentazione di funzioni

- **Annotazioni:** utili per documentare il tipo di param. e valore di ritorno (ma non c'è verifica!)
- **Docstring:** descrizione testuale di una funzione
- **help:** funzione per visualizzare la documentazione

```
def x_intercept(m: float, b: float) -> float:
    """
    Return the x intercept of the line y=m*x+b.
    The x intercept of a line is the point at which
    it crosses the x axis (y=0).
    ...
    return -b/m
```

PYTHON

Effetti collaterali

- Modifica di oggetti passati come parametri o variabili globali, operazioni di lettura/scrittura...
- Annnullano la **trasparenza referenziale**
 - Impossibile semplificare, sostituendo una chiamata a funzione col suo valore di ritorno (es. presenti operazioni di I/O)
- Rendono la funzione **non idempotente**
 - Chiamata più volte, con gli stessi parametri, restituisce risultati diversi
- → Difficile fare verifiche matematiche
 - $z = f(\sqrt{2}, \sqrt{2})$
 - $s = \sqrt{2}$
 - $z = f(s, s)$

Funzioni non idempotenti

- Esempio di semplificazione
 - $p = rq(x) + rq(y) * (rq(x) - rq(x))$
 - $p = rq(x) + rq(y) * (0)$
 - $p = rq(x) + 0$
 - $p = rq(x)$
- Ma se **rq** ha effetti collaterali, non si può!



```
base_value = 0 # global variable
```

PYTHON

```
def rq(x: int) -> int:  
    global base_value  
    base_value += 1  
    return x + base_value
```

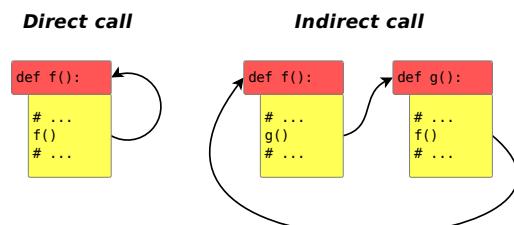
www.ce.unipr.it/people/tomamic

7/28

Programmazione ricorsiva

Programmazione ricorsiva

- Molti linguaggi consentono ad una funzione (o procedura) di chiamare se stessa
- Chiamata ricorsiva, diretta o indiretta



www.ce.unipr.it/people/tomamic

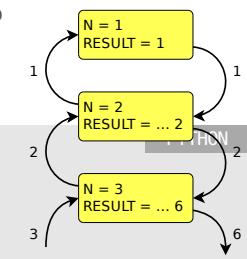
9/28

Fattoriale, ricorsione

- Ad ogni invocazione di una funzione, viene creato nello **stack** un nuovo record
- **Contesto locale** alla particolare attivazione della funzione stessa

```
def factorial(n: int) -> int:  
    result = 1  
    if n > 1:  
        result = n * factorial(n - 1)  
    return result
```

Ai primordi (Fortran 66 ecc.) solo allocazione statica
Spazio fisso ed unico per dati locali ad una funzione → no ricorsione

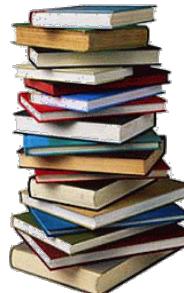


www.ce.unipr.it/people/tomamic

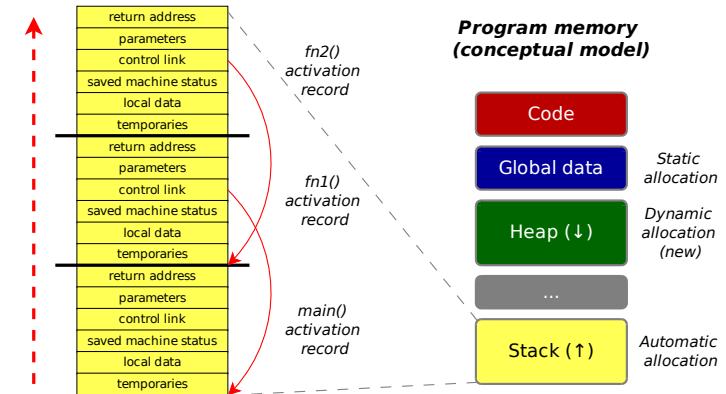
10/28

Stack dell'applicazione

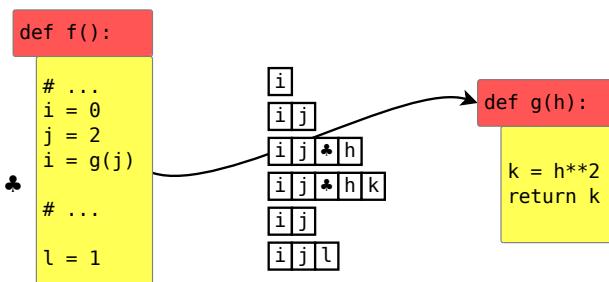
- Pila: memoria dinamica **LIFO (Last In First Out)**
 - Dimensione massima prefissata
- Il programma ci memorizza automaticamente:
 - Indirizzo di ritorno** per la funzione
Inserito alla chiamata, estratto all'uscita
 - Parametri** della funzione
Inseriti alla chiamata, eliminati all'uscita
 - Variabili locali**, definite nella funzione
Eliminate fuori dall'ambito di visibilità



Record di attivazione



Vista semplificata dello stack



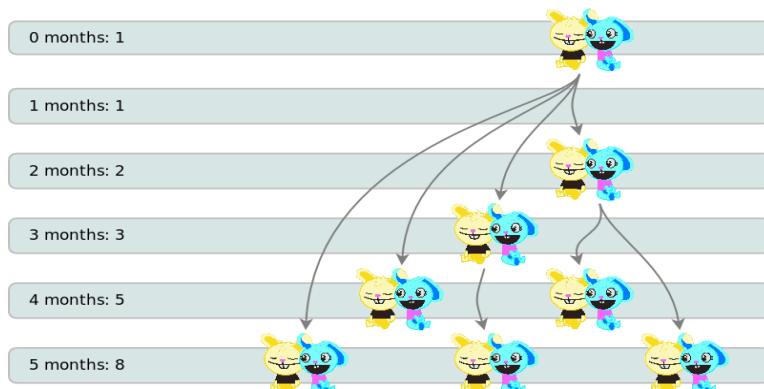
Visibilità di una variabile

- Insieme di istruzioni da cui è accessibile
 - Ciclo di vita**: esistenza in memoria della var (etichetta)
 - I valori (oggetti) in Python sono tutti gestiti dinamicamente
- Visibilità **globale**
 - Variabili fuori da ogni funzione - **Meglio evitare!**
 - Allocazione **statica** in alcuni linguaggi
- Visibilità **locale** alla funzione
 - Variabili locali e parametri
 - Allocazione **automatica** di spazio in **stack** ad ogni attivazione della funzione (possibile la ricorsione)
- Visibilità locale al blocco (es. **if**): non in Python!

I conigli di Fibonacci



Esempi di ricorsione



$fib(0) = fib(1) = 1; fib(n) = fib(n-1) + fib(n-2);$

www.ce.unipr.it/people/tomamic

16/28

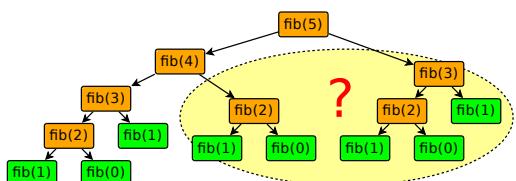
www.ce.unipr.it/people/tomamic

15/28

Fibonacci, ricorsione

```
def fibonacci(n: int) -> int:
    result = 1
    if n > 1:
        result = fibonacci(n-1) + fibonacci(n-2)
    return result
```

PYTHON



Fibonacci, memoization

```
_fibonacci_lookup = [1, 1]

def fibonacci(n: int) -> int:
    if n < len(_fibonacci_lookup):
        return _fibonacci_lookup[n]
    result = fibonacci(n - 1) + fibonacci(n - 2)
    _fibonacci_lookup.append(result)
    return result
```

PYTHON

```
from functools import lru_cache

@lru_cache()
def fibonacci(n: int) -> int:
    result = 1
    if n > 1:
        result = fibonacci(n-1) + fibonacci(n-2)
    return result
```

PYTHON

www.ce.unipr.it/people/tomamic

17/28

www.ce.unipr.it/people/tomamic

18/28

Fibonacci, iterazione

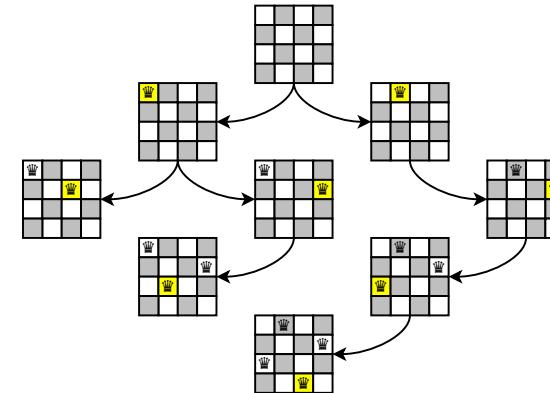
```
def fibonacci(n: int) -> int:
    value = 1
    previous = 0

    for i in range(n):
        value, previous = value + previous, value

    return value
```

PYTHON

N regine, backtracking



N regine, verifica

```
def print_board(board: list):
    for y in range(len(board)):
        for x in range(len(board)):
            if x == board[y]: print('|Q', end='')
            else: print('| ', end='')
        print()

def under_attack(board: list, x: int, y: int) -> bool:
    for i in range(y):
        d = y - i
        # directions: ↗↖ (no queens below)
        if board[i] in (x - d, x, x + d):
            return True
    return False
```

PYTHON

N regine, ricorsione

```
def place_queens(board: list, y=0) -> bool:
    if y == len(board):
        return True # all queens already placed
    for x in range(len(board)):
        if not under_attack(board, x, y):
            board[y] = x # (x, y) is safe: place a queen

            # try and place queens in the following rows
            if place_queens(board, y + 1):
                return True

            board[y] = None # no luck, backtrack
    return False
```

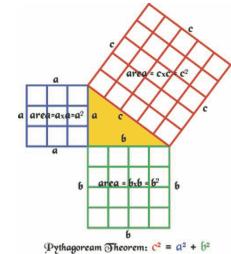
PYTHON

board è una lista di int: per ogni riga della scacchiera, memorizza la posizione x della regina



Ipotenusa

- Scrivere una funzione per il calcolo dell'ipotenusa di un triangolo rettangolo
 - Parametri: due cateti come float
 - Risultato: ipotenusa come float
- Nel **main**
 - Chiedere all'utente due valori,
 - Invocare la funzione con questi parametri
 - Visualizzare il risultato della funzione



Esercizi

Massimo Comun Divisore

- Leggere due numeri
- Calcolare in una funzione il loro Massimo Comun Divisore
- Visualizzare il risultato della funzione

Provare ad usare sia l'iterazione che la ricorsione

Euclide: $MCD(a, b) = a$, se $b = 0$;
 $MCD(a, b) = MCD(b, a \text{ mod } b)$, se $b > 0$



Torre di Hanoi

- Tre paletti + N dischi di diametro decrescente
- Portare tutti i dischi dal primo all'ultimo paletto
- Si può spostare solo un disco alla volta
- Non si può mettere un disco su uno più piccolo

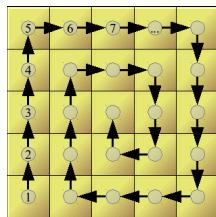
Usare la ricorsione. Immediato spostare un solo disco.
 N dischi: spostarne N-1 sul piolo né origine né dest.,
 spostare l'ultimo disco sul piolo giusto,
 spostare ancora gli altri N-1 dischi.



Spirale

- Scrivere una funzione per riempire di numeri crescenti una matrice quadrata (o rettangolare)
- Seguire il percorso a spirale suggerito nella figura a fianco
- Dimensioni della matrice indicate dall'utente a runtime

Tenere traccia della direzione attuale ($\Delta y, \Delta x$)
Avanzare fino al bordo o ad una cella già visitata,
poi cambiare la direzione in senso orario



Coordinate raster, rotazione oraria di 90°: $(x', y') = (-y, x)$
In generale: $(x', y') = (x \cdot \cos(\theta) - y \cdot \sin(\theta), x \cdot \sin(\theta) + y \cdot \cos(\theta))$



<Domande?>

Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR

Oggetti e grafica



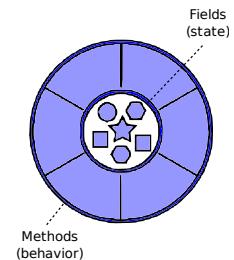
Introduzione alla
programmazione

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR



Oggetto

- Rappresenta un **oggetto fisico** o un **concetto** del dominio
- Memorizza il suo **stato** interno in **campi privati**
 - *Incapsulamento (black box)*
- Offre un insieme di **servizi**, come **metodi pubblici**
 - Realizza un *tipo di dato astratto (ADT)*



Classi ed oggetti

- Ogni **oggetto** ha una **classe** di origine
 - La classe dà la stessa forma iniziale (campi e metodi) a tutti i suoi oggetti
- Ma ogni **oggetto** ha la sua **identità**
 - Stato e locazione in memoria distinti da quelli di altri oggetti
 - Sia istanze di classi diverse che della stessa classe



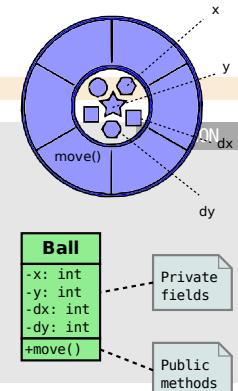
Definizione della classe

- **Incapsulamento** dei dati: *convenzione* sui nomi
 - Prefisso `_` per i nomi dei **campi privati**

“Siamo tutti adulti consenzienti. (GvR)”

```
class Ball:  
    W, H = 20, 20  
    ARENA_W, ARENA_H = 320, 240
```

```
    def __init__(self, x: int, y: int):  
        self._x = x  
        self._y = y  
        self._dx = 5  
        self._dy = 5  
    # ...
```



Class diagram UML

Costruzione oggetti

- **`__init__`**: metodo *inizializzatore*
 - Eseguito automaticamente alla creazione di un oggetto
 - *Instantiation is initialization*
- **`self`**: primo parametro di tutti i metodi
 - Non bisogna passare un valore esplicito
 - Assegnato l'oggetto di cui si chiama il metodo
 - Permette ai metodi di accedere ai campi
- Costanti definite direttamente nella **classe**
 - Per usarle, precedute dal nome della classe e `.`
 - Caratteristiche della classe, non di una singola istanza

```
ball = Ball(40, 80) # Allocation and initialization
```

PYTHON

Metodi

- Espongono **servizi** ad altri oggetti

```
class Ball:  
    # ...  
    def move(self):  
        if not (0 <= self._x + self._dx <= Ball.ARENA_W - Ball.W):  
            self._dx = -self._dx  
        if not (0 <= self._y + self._dy <= Ball.ARENA_H - Ball.H):  
            self._dy = -self._dy  
        self._x += self._dx  
        self._y += self._dy  
  
    def rect(self) -> (int, int, int, int):  
        return self._x, self._y, Ball.W, Ball.H
```

PYTHON

Applicazione

```
from ball import Ball # Ball is defined in ball.py

# Create two objects, instances of the Ball class
b1 = Ball(40, 80)
b2 = Ball(80, 40)
print('Ball 1 @', b1.rect())
print('Ball 2 @', b2.rect())

while input() != 'x':
    b1.move()
    b2.move()
    print('Ball 1 @', b1.rect())
    print('Ball 2 @', b2.rect())
```

PYTHON

Il primo parametro, self

- Il primo parametro di ogni metodo si chiama **self** (per convenzione)
- L'oggetto, di cui viene invocato il metodo, viene assegnato come valore di **self**
- In Python, una chiamata a metodo è interpretata così:

```
b1 = Ball(40, 80)
b1.move()
```

PYTHON

```
b1 = Ball(40, 80) # also, automatically call
                   # Ball.__init__(b1, 40, 80)
Ball.move(b1)
```

PYTHON

Nota. Meglio usare la prima notazione, che evidenzia l'oggetto anzichè la classe!

Proprietà

- Permettono un accesso controllato allo stato

```
class Ball:

    @property # a getter for the pos property
    def pos(self) -> (int, int):
        return self._x, self._y

    # @pos.setter # if you also really need a setter
    # def pos(self, val: (int, int)):
    #     self._x, self._y = val
```

PYTHON

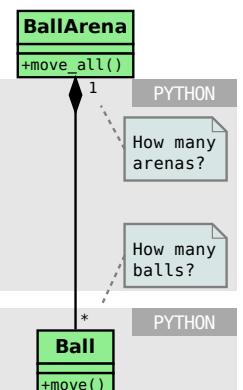
```
ball = Ball(40, 80)
print('ball @', ball.pos)
# ball.pos = (60, 20) # with the setter, you could change the pos
```

PYTHON

Composizione

- Associazione di tipo **has-a, part-of** tra oggetti
 - Una **arena** può **contenere** diverse **palline**

```
class BallArena: # ...
    def __init__(self):
        self._balls = []
    def add(self, b: Ball):
        self._balls.append(b)
    def move_all(self):
        for b in self._balls:
            b.move()
```



```
arena = BallArena()
arena.add(Ball(40, 80)); arena.add(Ball(80, 40)) # ...
arena.move_all()
```

Ciclo di vita di un oggetto

- Creazione di un oggetto: allocata memoria per tenere lo stato dell'oggetto
- In Python: variabile = riferimento ad un oggetto
 - Oggetti: *allocazione dinamica*, in memoria *heap*
 - Variabili: *allocazione automatica*, in memoria *stack*
- Oggetto non più associato a nessuna variabile: necessario liberare memoria
- **Garbage collection**: gestione automatica della restituzione di memoria

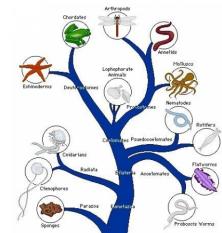


Garbage collection

- Vantaggi
 - Non è possibile dimenticare di liberare la memoria (*memory leak*)
 - Non è possibile liberare della memoria che dovrà essere utilizzata in seguito (*dangling pointer*)
- Svantaggi
 - Il garbage collector decide autonomamente quando liberare la memoria
 - Liberare e compattare la memoria richiede del calcolo
- Diversi algoritmi
 - *Reference counting*: idea di base, ma cicli...
 - *Mark & sweep*: parte da riferimenti locali/globali, marca oggetti raggiungibili
 - *Generational garbage collection*: controlla spesso oggetti recenti



Livelli di astrazione



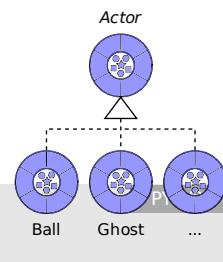
Livelli di astrazione

- Relazione **is-a** tra classi
 - Specializzazione, sotto-insieme
- Es. classificazioni in biologia
 - I *vertebrati* sono una sottoclasse degli *animali*
 - I *mammiferi* sono una sottoclasse dei *vertebrati*
 - I *felini* sono una sottoclasse dei *mammiferi*
 - I *gatti* sono una sottoclasse dei *felini*
- Ogni sottoclasse...
 - Eredita le caratteristiche della classe base
 - Ma introduce delle specializzazioni

Livelli di astrazione

- Actor: *classe base*
 - Dichiara un metodo `move` ecc.
- Vari attori: *classi derivate*
 - Ereditano caratteristiche di Actor
 - Definiscono comportamenti specifici

```
class Actor:  
    def move(self):  
        raise NotImplementedError("Abstract method")
```



Generalizzazione e riuso

```
class Arena: # ...  
    def __init__(self):  
        self._actors = []  
    def add(self, a: Actor):  
        self._actors.append(a)  
    def move_all(self):  
        for a in self._actors:  
            a.move()
```

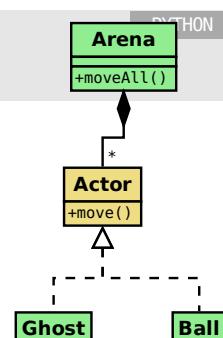
PYTHON

- Codice dipendente dalle classi più astratte, più in alto nella gerarchia
 - Arena riutilizzabile creando nuove classi derivate di Actor

Sostituzione

```
arena.add(Ball(40, 80))  
arena.add(Ghost(120, 40)) # ...  
arena.move_all()
```

- Principio di **sostituzione** di Liskov
 - Si può sempre usare un oggetto di una *classe derivata*, al posto di uno della *classe base*
- Relazione *has-a* tra un oggetto **Arena** e gli oggetti **Actor** che contiene
- Relazione *is-a* tra classi derivate (**Ball** e **Ghost**) e classe base (**Actor**)



Ereditarietà e polimorfismo

- **Classe derivata**
 - Eredita le caratteristiche della classe base
 - Può definire nuove caratteristiche specifiche
- **Metodo polimorfo**
 - Ridefinito nelle classi derivate
 - Attori diversi possono muoversi in modo diverso

```
class Ghost(Actor): # ...  
    def move(self):  
        dx = random.choice([-5, 0, 5])  
        dy = random.choice([-5, 0, 5])  
        arena_w, arena_h = self._arena.size()  
        self._x = (self._x + dx) % arena_w  
        self._y = (self._y + dy) % arena_h
```

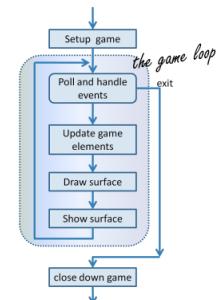
PYTHON



Grafica con pygame

- Libreria per giochi 2D
- Grafica e suoni
- Su *SDL* - Simple DirectMedia Layer
- Semplice e veloce
- Open-source
- Multi-piattaforma

pygame.org



Grafica con Pygame

Ciclo di animazione

```

import pygame
pygame.init()          # Prepare pygame
screen = pygame.display.set_mode((320, 240))
clock = pygame.time.Clock()      # To set game speed
image = pygame.image.load('ball.png')

x = 50; playing = True
while playing:
    for e in pygame.event.get(): # Handle events: mouse, keyb etc.
        if e.type == pygame.QUIT: playing = False
    screen.fill((255, 255, 255)) # Draw background
    screen.blit(image, (x, 50)) # Draw foreground
    x = (x + 5) % 320          # Update ball's position
    pygame.display.flip()       # Surface ready, show it!
    clock.tick(30)             # Wait 1/30 seconds
    pygame.quit()               # Close the window
  
```

PYTHON

Rimbalzi

```

arena = Arena(320, 240)
Ball(arena, 40, 80); Ball(arena, 80, 40);
Ghost(arena, 120, 80) # ...
# a map from an actor type to an image
images = {Ball: pygame.image.load('ball.png'),
          Ghost: pygame.image.load('ghost.png')}
screen = pygame.display.set_mode(arena.size())
playing = True
while playing:
    # Handle events here!

    arena.move_all()           # Game logic
    screen.fill((255, 255, 255)) # Background
    for a in arena.actors():
        x, y, w, h = a.rect()
        img = images[type(a)]
        screen.blit(img, (x, y)) # Foreground [...]
  
```



Tastiera e mouse

```
from pygame.locals import (KEYDOWN, KEYUP, K_RIGHT, K_d,
    MOUSEBUTTONDOWN, MOUSEBUTTONUP, MOUSEMOTION)
# ...
for e in pygame.event.get():
    # print(e)
    if e.type == KEYDOWN and e.key in (K_RIGHT, K_d):
        print('Right arrow (or D) pressed')
    elif e.type == KEYUP and e.key in (K_RIGHT, K_d):
        print('Right arrow (or D) released')
    elif e.type == MOUSEBUTTONDOWN and e.button == 1:
        print('Left mouse button pressed')
    elif e.type == MOUSEBUTTONUP and e.button == 1:
        print('Left mouse button released')
    elif e.type == MOUSEMOTION:
        print 'Mouse at (%d, %d)' % e.pos
```

PYTHON

Testo e suoni

```
# Red (anti-aliased) text, centered, rotated 30° ccw
font = pygame.font.SysFont('arial', 48)
surface = font.render('Game over!', True, (255, 0, 0))
surface = pygame.transform.rotate(surface, 30)
x = (screen.get_width() - surface.get_width()) // 2
y = (screen.get_height() - surface.get_height()) // 2
screen.blit(surface, (x, y)) # surface ~ image
```

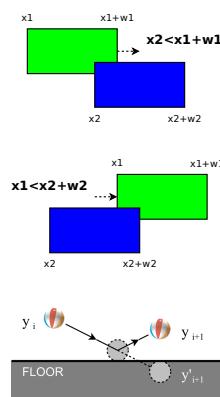
PYTHON

```
# Some sound
pick_up_sound = pygame.mixer.Sound('pickup.wav')
pick_up_sound.play() # play(-1) to loop, then stop()
```

PYTHON

Collisioni

- Molti algoritmi di *collision detection*
 - Casi semplici: intersezione di rettangoli
- In caso di collisione, Arena...
 - Invoca il metodo **collide** di entrambi gli oggetti
 - Collisione tra personaggio **self** e personaggio **other** (secondo parametro)
- Possibili errori nel calcolo del rimbalzo
 - Di solito accettabili
 - Altrimenti, applicare correzioni



Collaudo in Python

Come collaudare il codice?

- Usare un **debugger** per valutare espressioni in fase di esecuzione
 - Si può decidere cosa valutare a seconda del flusso di esecuzione e dei valori generati, senza ricompilare
- Istruzioni di **stampa** all'interno del programma
 - Valore di espressioni scritto a console o su file di log
- Entrambi gli stili, scarsamente **automatizzati**
 - Necessità di intervento attivo durante l'esecuzione dei test
 - Giudizio dei risultati da parte dell'utente
 - Quali valori analizzare? Sono coerenti?
- Scarsamente **componibili**
 - Difficile controllare molte espressioni nel debugger
 - "*Scroll blindness*": troppe istruzioni di stampa ⇒ codice poco leggibile

Libreria unittest

- I test **unittest** non richiedono continuo intervento o giudizio da parte dell'utente
- Facile eseguire molti test assieme, su un certo progetto
- Come definire un test?
 - Creare una sottoclasse di `unittest.TestCase`
 - Scrivere metodi di test, denominati con prefisso `test`
 - Per controllare la validità di una espressione, usare `assertTrue(bool)`

Esempio di test

- Controllare che una pallina rimbalzi correttamente contro il bordo inferiore

```
import unittest  
  
class SimpleBallTest(unittest.TestCase):  
  
    def test_bounce_down(self):  
        b = Ball(300, 220) # dx = 5, dy = 5  
        b.move()  
        self.assertTrue(b.rect() == (295, 215, 20, 20))  
  
if __name__ == '__main__':  
    unittest.main(exit=False)
```

PYTHON

Esecuzione dei test

- Meccanismi per definire i test da eseguire e organizzare i risultati
- Esecuzione di test dalla linea di comando
 - Inclusione dei test di un modulo, di una classe, o metodi di test specifici
 - Implementata anche una semplice forma di **test discovery**

```
python -m unittest test_module1 test_module2  
python -m unittest test_module.TestClass  
python -m unittest test_module.TestClass.test_method  
python -m unittest discover
```

CMD

- Annotazione `@unittest.skip("reason for skipping")`
 - Indica al framework di ignorare un certo metodo di test
 - Messaggio per documentare la decisione

Controllare le eccezioni

- Mestiere del programmatore
 - Codice che completa correttamente l'esecuzione nei casi normali...
 - Ma che anche in situazioni eccezionali mostra il comportamento atteso
- Come verificare che una eccezione attesa sia effettivamente sollevata?
 - Usare il metodo `assertRaises` direttamente, passando una funzione ed i parametri
 - Oppure creare un `contesto` con `with`
- Esempio: `Ball` solleva effettivamente una eccezione attesa?

```
def test_out_of_arena(self):  
    with self.assertRaises(ValueError):  
        b = Ball(-1, -1)
```

PYTHON

Test parametrizzati

- Ripetere un test con diversi parametri
 - Un test case per ogni gruppo di parametri?
 - In alcune applicazioni, enorme quantità di test!
- Soluzione semplicistica: test contente un ciclo
 - Ad ogni iterazione, preparato un gruppo di parametri diversi
 - Eseguite le istruzioni da testare sui nuovi parametri
 - Problema: il test si blocca al primo errore

www.ce.unipr.it/people/tomamic

31/38

www.ce.unipr.it/people/tomamic

32/38

Test parametrizzato, semplicistico

```
class ParamBallTest(unittest.TestCase):  
    TEST_VALUES = ( (40, 80, 45, 85),  
                    (40, 215, 45, 220),  
                    (40, 220, 45, 215),  
                    (295, 80, 300, 85),  
                    (300, 80, 290, 85) )  
  
    def test_move(self):  
        for param in ParamBallTest.TEST_VALUES:  
            x0, y0, x1, y1 = param  
            b = Ball(x0, y0)  
            b.move()  
            self.assertTrue(b.rect() == (x1, y1, 20, 20))
```

PYTHON

Sotto-test, Python 3.4

- Eseguiti tutti i sottotest, anche se uno fallisce

```
class ParamBallTest(unittest.TestCase):  
    TEST_VALUES = () # same values...  
  
    def test_move(self):  
        for param in ParamBallTest.TEST_VALUES:  
            with self.subTest(param=param):  
                x0, y0, x1, y1 = param  
                b = Ball(x0, y0)  
                b.move()  
                self.assertTrue(b.rect() == (x1, y1, 20, 20))
```

PYTHON

www.ce.unipr.it/people/tomamic

33/38

www.ce.unipr.it/people/tomamic

34/38

Fixture

- Due o più test operano su insiemi di oggetti uguali o simili
 - Questa configurazione iniziale comune si definisce **fixture**
- Se ci sono diversi test con una fixture comune...
 - Aggiungere dei campi per le varie parti della fixture
 - Inizializzare questi campi, nel metodo **setUp**
 - Liberare eventuali risorse allocate, nel metodo **tearDown**
- Una volta creata la fixture, può essere usata da tutti i test case
 - Aggiungere metodi di test alla classe
 - **setUp** e **tearDown** eseguiti prima e dopo ogni test

Esempio di fixture

- Numerosi metodi di test che operano su stessi dati iniziali
 - Esempio, una combinazione di palline in posizioni predefinite

```
class SimpleBallTest(unittest.TestCase):  
  
    def setUp(self):  
        self.b1 = Ball(80, 40)  
        self.b2 = Ball(40, 80)  
        self.b3 = Ball(120, 20)
```

PYTHON

Test suite

- Meccanismo per **raggruppare** logicamente dei test ed **eseguirli assieme**
- La classe **TestSuite** rappresenta una test suite
 - Lista di classi di test aggiunte con il metodo **addTest**
- La classe **TestRunner** rappresenta un esecutore di test
 - Per la console, si usa **TextTestRunner**, già inclusa nel framework

```
suite = unittest.TestSuite()  
suite.addTest(unittest.makeSuite(SimpleBallTest))  
runner = unittest.TextTestRunner()  
runner.run(suite)
```

PYTHON

<Domande?>



Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR



Linguaggio C++11



Introduzione alla programmazione

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Hello, C++

- `cout`: output su console, op. di inserimento `<<`
 - Possibile concatenare più operazioni di scrittura

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, C++!" << endl;
}
```

- Da *Qt Creator* (ambiente di sviluppo): *Create Project* → *Non-Qt* → *Plain C++*
- **C++11**: aggiungere al file `.pro` (di progetto): `CONFIG += c++11`

www.ce.unipr.it/people/tomamic

C++

2/40

Leggere e scrivere

- `cin`: input da console, op. di estrazione `>>`
 - Possibile concatenare più operazioni di lettura
 - `getline(cin, line)`: lettura intera riga

```
#include <iostream>
using namespace std;

int main() {
    string name;
    int age;
    cout << "Name, age?" << endl;
    cin >> name >> age;
    cout << "Hello, " << name << "." << endl;
    cout << "You're " << age << " years old." << endl;
}
```

C++

Tipizzazione statica

- Una delle differenze principali: le comuni variabili non sono *riferimenti*, ma *contenitori* di dati
 - Occorrono **dichiarazioni** di tipo
 - Ma possibile *type inference* (`auto`)
- Tipi principali: `int`, `float` (e `double`), `bool`
- `string`: sequenza mutabile di byte (tipo `char`)



```
int x = 10;
double h = 3.7;
string s = "hello";

auto y = 5;           // type inference: C++11
auto k = 2.2;
auto t = string{"hola"}; // C++14: auto t = "hola"s;
```

www.ce.unipr.it/people/tomamic

www.ce.unipr.it/people/tomamic

3/40

4/40

Operazioni di base

- Operazioni su numeri: +, -, *, /, %
- Anche incremento e decremento unitario: ++, --
- Attenzione:** la divisione tra interi dà risultato intero (`trunc`); il resto può essere negativo
- Assegnamento: =, +=, -= ...
- Confronti: >, >=, <, <=, !=, ==
- Attenzione:** i confronti **non** si possono concatenare
- Operazioni booleane (and, or, not): &&, ||, !

```
cout << (3 < 5) << endl;           // 1
cout << (3 < 5 < 4) << endl;         // 1 (!)
cout << (3 < 5 && 5 < 4) << endl; // 0
```

C++

Stringhe

- Operazioni di confronto; concatenazione: +
- Attenzione:** apici doppi per valori `string`, singoli per `char`

```
string sentence = "Lorem ipsum";
sentence[6] = 'I';
cout << sentence[6]; // 'I'

int n = 5;
string txt = to_string(n);
int val = stoi(txt); // see also 'stod', 'stof'...
```

C++

www.ce.unipr.it/people/tomamic

5/40

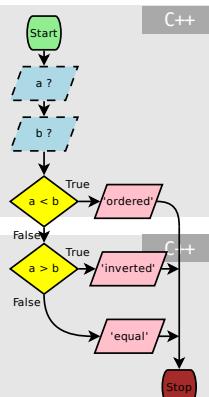
www.ce.unipr.it/people/tomamic

6/40

Decisioni

```
string a, b; cin >> a >> b;
if (a < b) {
    cout << "The words are ordered";
} else if (a > b) {
    cout << "The words are inverted";
} else {
    cout << "The words are equal";
}

int choice; cin >> choice;
switch (choice) {
    case 1: cout << "First option"; break;
    case 2: cout << "Second option"; break;
    default: cout << "Error";
}
```



C++

Iterazioni

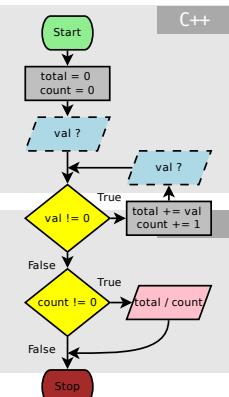
```
int val, tot = 0, count = 0;
cout << "Val (0 to end)? "; cin >> val;
while (val != 0) {
    tot += val; ++count;
    cout << "Val (0 to end)? "; cin >> val;
}
if (count > 0) cout << "Avg: " << tot / float(count);
```

www.ce.unipr.it/people/tomamic

7/40

www.ce.unipr.it/people/tomamic

8/40



C++

Vector, array dinamici

```
#include <vector>
// ...

vector<string> shopping_list = {"milk", "cocoa",
                                 "yogurt"};
cout << shopping_list[1] << endl; // cocoa
shopping_list[1] = "coffee";
shopping_list.push_back("corn flakes");
cout << shopping_list.size() << endl; // 4

shopping_list.erase(begin(shopping_list) + 2);
shopping_list.insert(begin(shopping_list) + 2, "biscuits");

vector<string> another_list;
another_list.assign(10, ""); // 10 strings
```



Cicli for

```
for (auto x : shopping_list) { // for-each, C++11
    cout << x << endl;
}

for (int i = 0; i < shopping_list.size(); ++i) { // range, C++98
    cout << shopping_list[i] << endl;
}
```

Somma colonne: matrice

```
vector<vector<int>> matrix = {{2, 4, 3, 8},
                                {9, 3, 2, 7},
                                {5, 6, 9, 1}};

auto rows = matrix.size();
auto cols = matrix[0].size();
for (auto x = 0; x < cols; ++x) {
    auto total = 0;
    for (auto y = 0; y < rows; ++y) {
        total += matrix[y][x];
    }
    cout << "Col #" << x << " sums to " << total << endl;
}

vector<vector<char>> another_matrix;
another_matrix.assign(rows, vector<char>(cols, '-'));
```

Funzioni

```
float cube(float x) {
    return x * x * x;
}

// pass by reference: external vars can be modified
void swap(int& m, int& n) {
    int tmp = m;
    m = n; n = tmp;
}

int main() {
    int a = 5, b = 7;
    swap(a, b);
    cout << a << " " << b << endl;
}
```

Flussi e file

```
#include <fstream>
// ...

int n; float r; string w;
ifstream file1{"input.txt"}; // file input stream
if (file1.good()) { // is stream available?
    file1 >> n >> r >> w;
}
file1.close();

ofstream file2{"output.txt"}; // file output stream
if (file2.good()) { // is stream available?
    file2 << "Values: " << n << " " << r << " " << w << endl;
}
file2.close();
```

C++

Lettura di righe

```
ifstream file1{"input.txt"}; // file input stream
```

C++

```
string first_line, second_line;
getline(file1, first_line);
getline(file1, second_line);
// read lines do not include newline
```

C++

```
string whole_text;
getline(file1, whole_text, '\0');
```

C++

```
string line;
while (getline(file1, line)) {
    cout << line << endl;
}
```

C++

Lettura di dati

```
int val; // or float, string ...
while (file1 >> val) {
    cout << setw(4) << val << endl; // val occupies 4 chars
} // setw in <iomanip>
```

C++

```
char val;
file1 >> noskipws; // otherwise, whitespaces are skipped
while (file1 >> val) {
    cout << val << endl;
}
```

C++

```
int n; float r; string w;
istringstream sstr{"5 7.5 hello"}; // istringstream in <sstream>
sstr >> n >> r >> w; // a stream view on a string
```

C++

Flussi e stringhe

- Si può gestire una stringa come uno stream
 - **istringstream, ostringstream** in libreria **<sstream>**
 - Per estrarre valori ed inserire valori, rispettivamente

```
/* Split a text into a vector of strings */
string text = "one:two::three";
vector<string> values;

istringstream sstr{text};
string item;
while (getline(sstr, item, ':')) {
    values.push_back(item);
}
```

C++

Numeri casuali

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
    srand(time(nullptr));           // just once! (initial seed
                                    // for random numbers)

    for (int i = 0; i < 10; ++i) { // generate 10 random numbers
        int r = rand() % 90; // 0 <= r < 90
        cout << r << endl;
    }
}
```

C++11 ha anche una libreria `random`, più avanzata
<http://www.stroustrup.com/C%2B%2BFAQ.html#std-random>

C++

Rettangoli e cerchi con Qt

```
#include <QtWidgets>

QPixmap screen{600, 400}; QPainter painter(&screen);

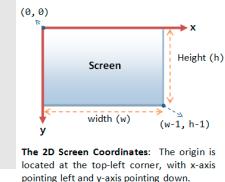
painter.setBrush(QColor{255, 255, 0});
painter.drawRect(50, 75, 90, 50);

painter.setBrush(QColor{0, 0, 255});
painter.drawEllipse(QPoint{300, 50}, 20, 20);

QLabel label; label.setPixmap(screen); label.show();
```



Code less.
Create more.
Deploy everywhere.



PROJECT

QT += widgets
CONFIG += console c++11

Progetto: Application → Qt Widgets Application

www.ce.unipr.it/people/tomamic

18/40

www.ce.unipr.it/people/tomamic

17/40



Oggetti

Oggetti

- C++: definizione della classe separata dalla implementazione dei metodi
 - Definizione fornita agli utenti
 - Implementazione compilata in libreria
- Sorgenti organizzati in 3 file:
 - `ball.h` – definizione della classe
 - `ball.cpp` – implementazione dei metodi
 - `main.cpp` – applicazione che usa la classe
 - Dall'ambiente di sviluppo: `Add new → C++ Class`

www.ce.unipr.it/people/tomamic

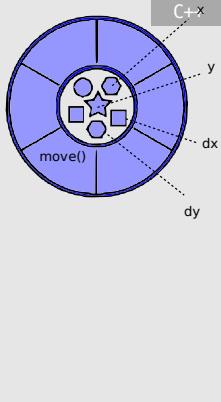
20/40

www.ce.unipr.it/people/tomamic

19/40

Definizione: ball.h

```
class Ball {  
public:  
    Ball(int x0, int y0);  
    void move();  
    int get_x();  
    int get_y();  
  
    static const int ARENA_W = 320;  
    static const int ARENA_H = 240;  
    static const int W = 20;  
    static const int H = 20;  
  
private:  
    int x; int y;  
    int dx; int dy;  
};
```



Implementazione: ball.cpp

```
#include "ball.h"  
  
Ball::Ball(int x0, int y0) {  
    x = x0; y = y0; dx = 5; dy = 5;  
}  
void Ball::move() {  
    if (!(0 <= x + dx && x + dx <= ARENA_W - W)) dx = -dx;  
    if (!(0 <= y + dy && y + dy <= ARENA_H - H)) dy = -dy;  
    x += dx; y += dy;  
}  
int Ball::get_x() {  
    return x;  
}  
int Ball::get_y() {  
    return y;  
}
```

Applicazione: main.cpp

```
#include "ball.h"  
// ...  
int main() {  
    Ball ball1{40, 80};  
    Ball ball2{80, 40};  
  
    string line;  
    while (getline(cin, line)) {  
        ball1.move();  
        ball2.move();  
  
        cout << ball1.get_x() << ", " << ball1.get_y() << endl;  
        cout << ball2.get_x() << ", " << ball2.get_y() << endl << endl;  
    }  
}
```

Allocazione dinamica

```
// ...  
int main() {  
    Ball ball1{40, 80};  
    Ball* ball2 = new Ball{80, 40};  
    // Ball* alias1 = &ball1; // no new ball is created  
    // Ball* alias2 = ball2; // no new ball is created  
  
    for (string line; getline(cin, line);) {  
        ball1.move();  
        ball2->move();  
        cout << ball1.get_x() << ", " << ball1.get_y() << endl;  
        cout << ball2->get_x() << ", " << ball2->get_y() << endl << endl;  
    }  
    delete ball2;  
}
```

Swig: C++ per moduli Python

```
%module ball
%include "std_string.i"
%{
#include "ball.h"
%}
%include "ball.h"
```

```
swig -python -c++ ball.i
g++ -fPIC -c ball.cpp ball_wrap.cxx -I/usr/include/python3.4/ -std=c++11
g++ -shared ball.o ball_wrap.o -o _ball.so
```

```
>>> from ball import Ball
>>> b = Ball(60, 60)
>>> b.move()
>>> print(b.get_x(), b.get_y())
```

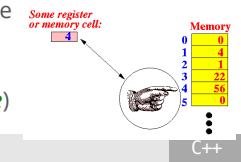
FILE: BALL.I

CMD

PYTHON

Puntatori

- Ogni dato presente in memoria ha un indirizzo: variabile puntatore per memorizzarlo
 - Operatore & per indirizzo di un dato
 - Op. * per accesso a dato puntato (*dereferenziazione*)



```
char i = 56; // a byte
char* p; // a ptr to some byte (uninitialized)
p = &i; // now p points to i
*p = *p + 1; // ++i
p = nullptr; // ptr to nothing
```

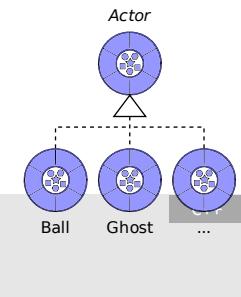
- No *garbage collection*: a **new** deve corrispondere **delete**
- Resource Acquisition Is Initialization (RAII)*
 - Costruttore* alloca risorse, *distruttore* le libera: ~**Ball()**

Livelli di astrazione



Livelli di astrazione

- Actor:** *classe base*
 - Dichiara un metodo **move** ecc.
 - virtual**: il metodo può essere ridefinito nelle sottoclassi (*polimorfo*)
 - = **0**: il metodo non è implementato qui (la classe è *astratta*)



```
class Actor {
    virtual void move() = 0;
    // ...
};
```

Composizione

```
class Arena { // ...
public:
    void add(Actor* c);
    void move_all();
private:
    vector<Actor*> actors;
};
```

```
void Arena::add(Actor* c) {
    actors.push_back(c);
}
void Arena::move_all() {
    for (auto c : actors) c->move();
}
```

C++

C++

Ereditarietà e polimorfismo

```
arena->add(new Ball(4, 8));
arena->add(new Ghost(12,4));
arena->move_all();
```

```
class Ghost: public Actor {
// ...
Ghost() { /* ... */ }
```

```
void move() {
    vector<int> vals = {-5, 0, 5};
    int dx = vals[rand() % 3]; // one of {-5, 0, +5}
    int dy = vals[rand() % 3];
    x = (x + dx + arena->get_w()) % arena->get_w();
    y = (y + dy + arena->get_h()) % arena->get_h();
}
```

C++

C++

Ciclo di animazione in Qt

```
// fields: int x = 0; QPixmap image{"ball.png"};
Widget::Widget() {
    startTimer(1000 / 60); // 60 fps
}
void Widget::timerEvent(QTimerEvent* event) {
    x = (x + 5) % 600; // or width()
    update(); // async: this widget should be redrawn
}
void Widget::paintEvent(QPaintEvent* event) {
    QPainter painter{this};
    painter.drawPixmap(x, 10, image);
}
```

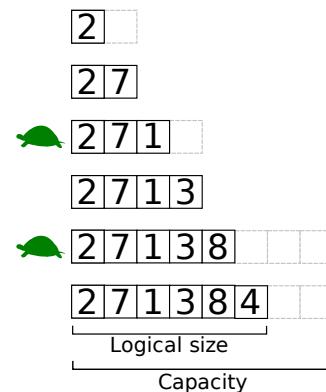
C++



Strutture dati lineari

Vector, array dinamici

- **Array**: area di memoria che contiene in *celle contigue* elementi tutti dello *stesso tipo*
- Usato internamente dai **vector** del C++
 - Riallocazione dinamica e trasparente per inserimenti e rimozioni
- **Vector** come **array dinamici**
 - Accesso casuale: $O(1)$
 - Aggiunta o rimozione in fondo all'array: $O(1)$, ma a volte riallocazione
 - Inserimento o rimozione: $O(n)$



Vector di float

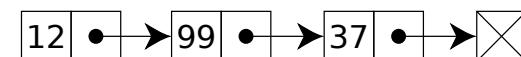
```
C++  
class FloatVector { // ...  
    int capacity_;  
    int size_;  
    float* data_;  
public:  
    FloatVector(int size, float val);  
    float get(int pos);  
    void set(int pos, float val);  
    void insert(int pos, float val);  
    float remove(int pos);  
    int size();  
};
```

Implementazione nel repository di esempi

Inserimento in vector

```
C++  
void FloatVector::insert(int pos, float val) {  
    if (pos < 0 || pos > size_) throw out_of_range("wrong pos");  
    if (size_ == capacity_) expand_capacity();  
    for (int i = size_; i > pos; --i) data_[i] = data_[i - 1];  
    data_[pos] = val;  
    ++size_;  
}  
void FloatVector::expand_capacity() {  
    capacity_ *= 2;  
    float* bigger = new float[capacity_];  
    for (int i = 0; i < size_; i++) bigger[i] = data_[i];  
    delete[] data_;  
    data_ = bigger;  
}
```

Liste concatenate



- Ciascun **nodo** contiene un *valore* della lista ed un *puntatore* al nodo successivo
 - Accesso casuale: $O(n)$
 - Aggiunta o rimozione in testa all'array: $O(1)$
 - Aggiunta o rimozione in fondo all'array: $O(n)$, oppure $O(1)$ se noto ultimo nodo
 - Inserimento o rimozione: $O(1) + O(n)$ per ricerca

Lista di float

```
struct Node {  
    float val;  
    Node* next;  
};  
class FloatList { // ...  
    Node* head_; int size_;  
public:  
    FloatList(int size, float val);  
    float get(int pos);  
    void set(int pos, float val);  
    void insert(int pos, float val);  
    float remove(int pos);  
    int size();  
};
```

Implementazione nel repository di esempi

 www.ce.unipr.it/people/tomamic

C++

Inserimento in lista

```
void FloatList::insert(int pos, float val) {  
    if (pos < 0 || pos > size_) throw out_of_range("wrong pos");  
    if (pos == 0) {  
        head_ = new Node{val, head_};  
    } else {  
        Node* n = head_;  
        for (int i = 0; i < pos - 1; ++i) n = n->next;  
        n->next = new Node{val, n->next};  
    }  
    ++size_;
```

C++

Template C++

- **Programmazione generica:** codice che opera su *tipi parametrizzati*
 - Es. liste di `int`, `string` ecc.: cambia solo il tipo di dato!
- **Template:** meccanismo di programmazione generica in C++
 - Generaz. trasparente di codice specializzato per un tipo
 - Es. `vector<float>` genera una classe ≈ `FloatVector`

```
template <class T>  
T max(T a, T b) {  
    if (a >= b) return a;  
    return b;  
}
```

```
double x = max<double>(5.0, 3.5); # <double>, <int>: optional  
int i = max<int>(4, 6); # inferred from parameters
```

37/40

 www.ce.unipr.it/people/tomamic

38/40

 www.ce.unipr.it/people/tomamic

C++

<Domande?>



Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR

39/40



Gui con Qt



Introduzione alla programmazione

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Caratteristiche di Qt

Sviluppo con meno codice

- *Design OO*: classi intuitive, riusabili ed estendibili
- Sviluppo *visuale* o *dichiarativo* di gui
- Personalizzazione aspetto delle app con *CSS*

Sviluppo libero e multipiattaforma

- *Qt Project*: codice open source (Nokia → Digia)
- *Qt Creator*: ambiente integrato di sviluppo
- **App portabili e avanzate**: buone prestazioni con poche risorse
 - Sistemi desktop: Windows, MacOS, Linux ...
 - Mobile/embedded: Android, iOS, Symbian, BlackBerry QNX, MeeGo-JollaOS, Tizen, Ubuntu Touch, Raspberry Pi ...
 - KDE, KOffice, VLC, Google Earth, Skype (Linux), Mathematica...

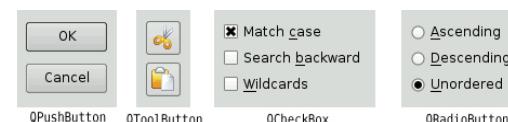
www.ce.unipr.it/people/tomamic

2/58

Libreria modulare

- Qt Core: classi base, stringhe ecc.
- **Qt GUI**: supporto grafica 2D
 - Integraz. OpenGL, anti-aliasing, trasparenza, trasformaz. vettoriali
- **Qt Widgets**: insieme di *widget* evoluti
 - Usabilità, esperienza più soddisfacente per utente
- Qt Multimedia, Qt Multimedia Widgets
- Qt Network
- Qt QML, Qt Quick, Qt Quick Controls, Qt Quick Layouts
- Qt SQL
- Qt Test
- Qt WebKit, Qt WebKit Widgets

Bottoni e display

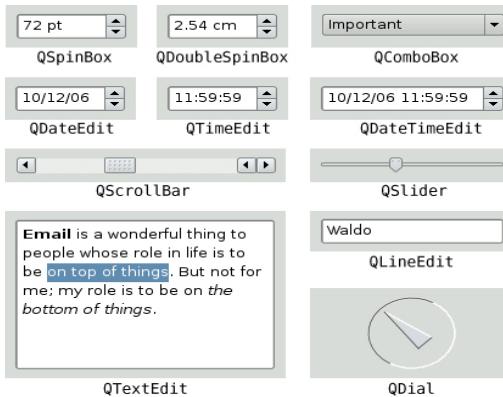


www.ce.unipr.it/people/tomamic

3/58

4/58

Input

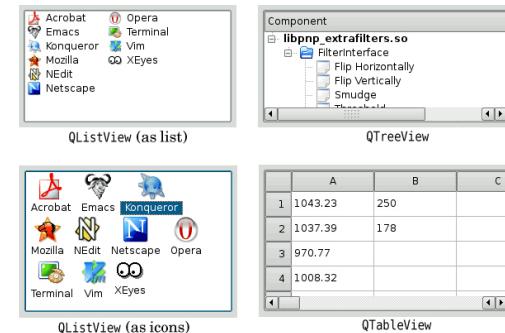


 www.ce.unipr.it/people/tomamic



Esempio, con
QCalendarWidget

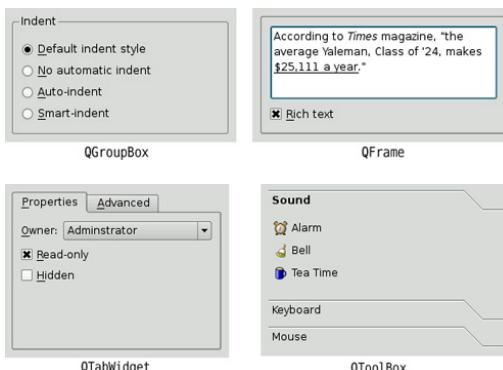
Viste di elementi



 www.ce.unipr.it/people/tomamic

5/58

Contenitori

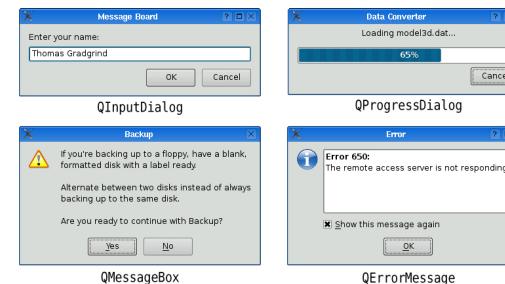


Inoltre: QMenu, QMenuBar, QToolBar, QStatusBar

 www.ce.unipr.it/people/tomamic

5/58

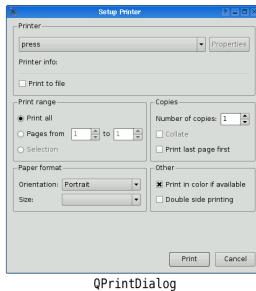
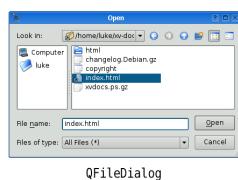
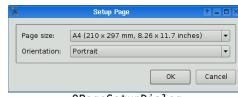
Finestre di dialogo



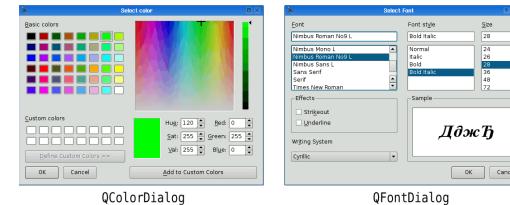
 www.ce.unipr.it/people/tomamic

6/58

File e stampa



Colori e font



Stili

- Qt sfrutta le primitive grafiche della piattaforma
 - Efficienza, aspetto familiare
 - Ma possibile stile personalizzato!
 - `QApplication::setStyle(new QWindowsStyle);`
 - `QApplication::setStyleSheet, QWidget::setStyleSheet`



DII

Disposizione dei widget

Visualizzare un widget

```
#include <QApplication>
#include <QTextEdit>

int main(int argc, char* argv[]) {
    QApplication app{argc, argv};

    QTextEdit text_edit;
    text_edit.show();

    return app.exec(); // event management
}
```



Creare un nuovo widget

- Estendere **QWidget**, o una sua sottoclasse
 - Creare nuovi campi e metodi, sovrascrivere metodi **virtual**
 - Altrimenti, **ereditate** le caratteristiche della classe base
- Incapsulare parti dell'interfaccia utente
 - Nuovo widget: **composto** da widget elementari
 - Resto dell'applicazione non ha bisogno di conoscere i dettagli
- Nuovo widget riusabile
 - Nella stessa applicazione o in altri progetti

Sottoclasse di QWidget

```
class Notepad : public QWidget {
public:
    Notepad();

private:
    QTextEdit* text_edit; // simple widgets;
    QPushButton* exit_button; // encapsulated as private fields
};
```



- Da **Qt Creator**:
 - *Create Project → Applications → Qt Gui Application*
 - **Base class:** QWidget -- **Generate form:** no
- **C++11**: aggiungere al file .pro (di progetto): **CONFIG += C++11**

Costruire la GUI

```
Notepad::Notepad() {
    // construtor: build the GUI
    // QObject::tr translates GUI texts (see Qt Linguist)

    text_edit = new QTextEdit;
    exit_button = new QPushButton{tr("Quit")};

    // widgets in a vertical layout
    auto layout = new QVBoxLayout;
    layout->addWidget(text_edit);
    layout->addWidget(exit_button);

    setLayout(layout); // QWidget
}
```



Layout principali

- Qt usa il meccanismo dei *layout* per disporre i widget
- Ci sono tre classi di layout principali
 - QHBoxLayout
 - QVBoxLayout
 - QGridLayout
- Per installare un layout su un widget, dobbiamo invocare il metodo `setLayout` del widget
- Il layout gestisce l'area *interna* al widget

Layout compositi

- È possibile inserire un layout dentro un altro
- Es. Layout aggiuntivo a destra, con bottoni in verticale

```
// ctor
auto button_layout = new QVBoxLayout;
button_layout->addWidget(open_button);
button_layout->addWidget(save_button);
button_layout->addWidget(exit_button);
button_layout->addStretch();

auto main_layout = new QHBoxLayout;
main_layout->addWidget(text_edit);
main_layout->addLayout(button_layout);
setLayout(main_layout);
```

With and without
addStretch()

Click dei buttoni

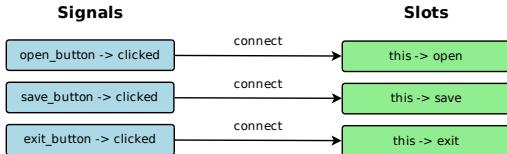
Definire gli slot



```
class Notepad : public QWidget {
    // add support for signals and slots...
    Q_OBJECT
public:
    Notepad();
public slots: // special methods, to connect with signals
    void open();
    void save();
    void exit();
private:
    QTextEdit* text_edit;
    QPushButton* open_button;
    QPushButton* save_button;
    QPushButton* exit_button;
};
```

Connettere segnali e slot

```
C++  
Notepad::Notepad() {  
    // ... at the end of ctor ...  
    connect(open_button, &QPushButton::clicked, this, &Notepad::open);  
    connect(save_button, &QPushButton::clicked, this, &Notepad::save);  
    connect(exit_button, &QPushButton::clicked, this, &Notepad::exit);  
}
```



Accoppiamento tra segnali e slot

Accoppiamento lasco

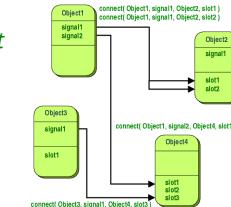
- Un oggetto emette un **segnaile**, ma non sa quali **slot** lo ricevono
- Molti segnali ad un singolo slot, un segnale a molti slot

Type safe

- La **firma** del segnale (numero e tipo di parametri) deve corrispondere a quella degli slot collegati
- Se la firma di uno slot è più corta, trascura degli argomenti che riceve
- Compilatore rileva errori

Estensione alla sintassi C++

- Segnali e slot sono una estensione specifica di Qt alla sintassi del C++



Aprire un file

```
C++  
void Notepad::open() {  
    // choose the input file  
    auto filename = QFileDialog::getOpenFileName(this);  
    if (filename != "") {  
        ifstream in{filename.toStdString()};  
        if (in.good()) {  
            // read the whole text  
            string content; getline(in, content, '\0');  
            text_edit->setText(content.c_str());  
        } else {  
            QMessageBox::critical(this, tr("Error"),  
                                 tr("Could not open file"));  
        }  
    }  
}
```

Salvare in un file

```
C++  
void Notepad::save() {  
    // choose the output file  
    auto filename = QFileDialog::getSaveFileName(this);  
    if (filename != "") {  
        ofstream out{filename.toStdString()};  
        if (out.good()) {  
            // write the whole text  
            auto text = text_edit->toPlainText();  
            out << text.toStdString();  
        } else {  
            QMessageBox::critical(this, tr("Error"),  
                                 tr("Could not save file"));  
        }  
    }  
}
```

Chiudere l'app

```
void Notepad::exit() {
    auto button = QMessageBox::question(
        this,
        tr("Notepad - Quit"),
        tr("Do you really want to quit?"),
        QMessageBox::Yes | QMessageBox::No);

    if (button == QMessageBox::Yes) {
        window()->close();
    }
}

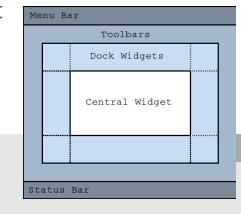
// See documentation (F1): QMessageBox, QInputDialog, QDialog
```

C++

Finestra principale

- **QMainWindow**: widget complesso, con un proprio layout particolare, per aggiungere:
 - QMenuBar, QStatusBar, QToolBar, QDockWidget
 - Widget principale al centro

```
// ... set a layout in the central area
setCentralWidget(new QWidget);
centralWidget()->setLayout(layout);
```



 www.ce.unipr.it/people/tomamic

25/58

 www.ce.unipr.it/people/tomamic

26/58

Menù e toolbar

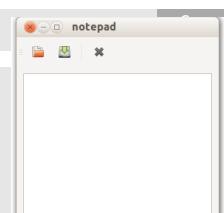
```
class NotepadWindow: public QMainWindow // ...

NotepadWindow::NotepadWindow() {
    auto notepad = new Notepad; setCentralWidget(notepad);

    auto menu = menuBar()->addMenu(tr("&File")); // QMenu*
    auto open_act = menu->addAction(tr("&Open")); // QAction*
    auto save_act = menu->addAction(tr("&Save"));
    menu->addSeparator();
    auto exit_act = menu->addAction(tr("E&xit"));

    // auto tools = addToolBar(tr("&File")); tools->addAction(open_act); ...

    connect(open_act, &QAction::triggered, notepad, &Notepad::open);
    connect(save_act, &QAction::triggered, notepad, &Notepad::save);
    connect(exit_act, &QAction::triggered, notepad, &Notepad::exit);
}
```



Griglia di bottoni

 www.ce.unipr.it/people/tomamic

27/58

 www.ce.unipr.it/people/tomamic

28/58

Griglia di bottoni

- **QGridLayout**: dispone i widget in una griglia
- All'inserimento del widget, specificare riga e colonna (**0-indexed**)
- Possibile specificare anche l'occupazione di più celle adiacenti



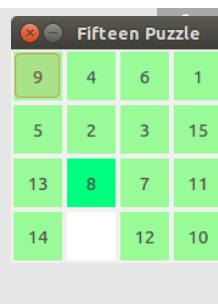
Fifteen – Gioco astratto

```
class Game {  
public:  
    virtual void play_at(int x, int y) = 0;  
    virtual int cols() const = 0;  
    virtual int rows() const = 0;  
    virtual std::string get_val(int x, int y) const = 0;  
    virtual bool finished() const = 0;  
    virtual std::string message() const = 0;  
};
```



Fifteen – Gui generica

```
class GameGui : public QWidget {  
    Q_OBJECT  
public:  
    GameGui(Game* game);  
private:  
    void handle_click(int x, int y);  
    void update_button(int x, int y);  
    void update_all_buttons();  
  
    Game* game_;  
    int cols_, rows_;  
};
```



Fifteen – Costruzione gui

```
GameGui::GameGui(Game* game) {  
    cols_ = game.cols(); rows_ = game.rows();  
    game_ = game;  
    auto grid = new QGridLayout; setLayout(grid);  
    for (auto y = 0; y < rows_; ++y) {  
        for (auto x = 0; x < cols_; ++x) {  
            auto b = new QPushButton;  
            grid->addWidget(b, y, x);  
            connect(b, &QPushButton::clicked,  
                    [=]{ handle_click(x, y); });  
        }  
    }  
    update_all_buttons();  
}
```

- **Funzioni lambda**: anonime, annidate (*Church*, 1936)
- **Closure**: cattura di valori dal contesto; **this**, **x**, **y**

Fifteen – Aggiornamento bottoni

```
void GameGui::update_button(int x, int y) {
    auto val = game_->get_val(x, y);
    auto b = layout()->itemAt(y * cols_ + x)->widget();
    dynamic_cast<QPushButton*>(b)->setText(val.c_str());
}

void GameGui::update_all_buttons() {
    for (auto y = 0; y < rows_; y++) {
        for (auto x = 0; x < cols_; x++) {
            update_button(x, y);
        }
    }
}
```

C++

Fifteen – Gestione click

```
void GameGui::handle_click(int x, int y) {
    game_->play_at(x, y);
    update_all_buttons(); // ...

    if (game_->finished()) {
        QMessageBox::information(this,
                               tr("Game finished"),
                               tr(game_->message().c_str()));
        window()->close();
    }
}
```



Pyside



Pyside – Costruzione gui

```
class GameGui(QWidget):
    def __init__(self, game: Game):
        QWidget.__init__(self)
        self._game = game
        self._cols, self._rows = game.size()
        self.setLayout(QGridLayout())
        for y in range(self._rows):
            for x in range(self._cols):
                b = QPushButton()
                self.layout().addWidget(b, y, x)
                b.clicked.connect(lambda x=x, y=y:
                                 self.handle_click(x, y))
        self.update_all_buttons()
    # ...
```



Pyside - Aggiornamento bottoni

```
class GameGui(QWidget):  
    # ...  
    def update_button(self, x: int, y: int):  
        val = self._game.get_val(x, y)  
        b = self.layout().itemAt(y * self._cols + x).widget()  
        b.setText(val)  
  
    def update_all_buttons(self):  
        for y in range(self._rows):  
            for x in range(self._cols):  
                self.update_button(x, y)
```

PYTHON

Pyside - Gestione click

```
class GameGui(QWidget):  
    # ...  
    def handle_click(self, x: int, y: int):  
        self._game.play_at(x, y)  
        self.update_all_buttons()  
  
    if self._game.finished():  
        QMessageBox.information(self,  
                                QMessagebox.information(self,  
                                self.tr('Game finished'),  
                                self.tr(self._game.message()))  
        self.window().close()
```



Eventi in Qt

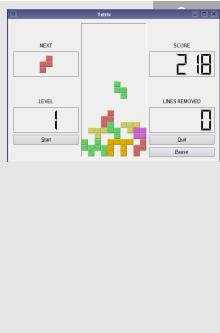


Dispatching degli eventi

- Eventi
 - Attività *esterne* che interessano l'applicazione
 - O cambiamenti *interni* all'applicazione
 - Gestibili da qualsiasi istanza di **QObject** (spesso widget)
- Quando si verifica un evento:
 - Creato oggetto per rappresentarlo, istanza (indiretta) di **QEvent**
 - **QResizeEvent**, **QPaintEvent**, **QMouseEvent**, **QKeyEvent**, **QCloseEvent**
- Sulla base del **tipo** di evento:
 - Invocato un **metodo specifico** dell'oggetto interessato
 - Dispatching attraverso il metodo **event** di **QObject**
 - L'evento può essere accettato oppure ignorato

Eventi della tastiera

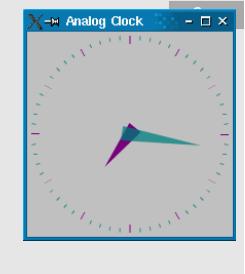
```
void TetrixBoard::keyPressEvent(QKeyEvent* e) {  
    switch (e->key()) {  
        case Qt::Key_Left:  
            tryMove(curPiece, curX - 1, curY);  
            break;  
        // ...  
        case Qt::Key_Down:  
            tryMove(curPiece.rotatedRight(), curX, curY);  
            break;  
        default:  
            QFrame::keyPressEvent(e);  
    }  
}
```



Metodi `keyPressEvent` e `keyReleaseEvent` per gestire la tastiera

Segnali periodici

```
AnalogClock::AnalogClock() {  
    QTimer* timer = new QTimer(this);  
    connect(timer, &QTimer::timeout,  
            this, &AnalogClock::update);  
    timer->start(1000);  
}  
  
void AnalogClock::paintEvent(QPaintEvent *event) {  
    QPainter painter[this];  
    // draw the clock ...  
}
```



`QTimer`: periodicamente emette segnale `timeout`, da associare ad uno (o più) slot

Metodo `paintEvent` per ridisegno di un widget, con oggetto `painter`

Disegni e animazioni

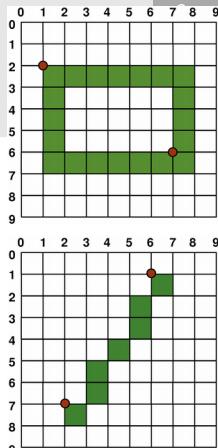


Ridisegno

- Metodo `update` accoda una richiesta di ridisegno *asincrono* del widget (evento)
 - L'operazione non avviene immediatamente
 - L'`event dispatcher` esegue il metodo `paintEvent` appena può
- Widget Qt operano di default in *double buffering*
 - No *flicker* dovuto a lettura e scrittura effettuate direttamente su aree di memoria → schermo
 - Es. si traccia sfondo prima di img in primo piano
 - Se parte visualizzazione a schermo, img in primo piano scompare per un frame

Sistema di coordinate

```
// ...
QPainter painter{this};
painter.setPen(Qt::darkGreen);
painter.drawRect(1, 2, 6, 4);
painter.drawLine(2, 7, 6, 1);
```



45/58

Disegno di immagini

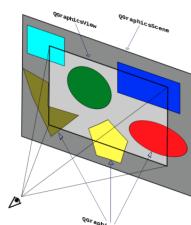
- **QPainter** può disegnare anche immagini **QPixmap**
 - **QPainter::drawPixmap(int x, int y, const QPixmap& pixmap);**
 - Immagine caricata facilmente da file, passandone il nome al costruttore
- Immagini su bottoni ed etichette
 - **QPushButton::setPixmap(QPixmap& pixmap)**
 - **QLabel::setIcon(QIcon& icon)**, oppure **setText** con HTML
 - Altri elementi, sfondo: **QWidget::setStyleSheet**
- **QPixmap** sono anche superfici di disegno custom
 - **QPainter::QPainter{QPaintDevice* d};**

www.ce.unipr.it/people/tomamic

46/58

Superfici ed elementi grafici

- **QGraphicsScene**
 - Superficie che contiene diversi elementi grafici bidimensionali: fornisce supporto per animazioni e rilevamento collisioni
- **QGraphicsItem**
 - Immagini, linee, poligoni, testo e altri elementi
- **QGraphicsView**
 - Widget per visualizzare l'intera scena o per zoomare su una parte



47/58



Dichiarazione di segnali

www.ce.unipr.it/people/tomamic

48/58

Dichiarazione di segnali

- Qt: distribuzione dei segnali gestita automaticamente
 - I segnali si dichiarano in maniera simile a metodi
 - Non devono essere implementati da nessuna parte
 - Devono restituire **void**
- Nota sugli argomenti
 - L'esperienza mostra che segnali e slot sono più riusabili se non usano tipi speciali
 - Raccogliere segnali da diversi widget sarebbe molto più difficile

Emissione di segnali

- Un oggetto emette un segnale chiamando **emit**
 - Quando si verifica evento o stato interno cambia
 - Se il cambiamento può interessare altri oggetti
- Emesso segnale → slot connessi eseguiti subito
 - Come una normale chiamata a metodo
 - Codice seguente ad **emit** eseguito dopo aver eseguito tutti gli slot connessi al segnale
 - Se più slot, eseguiti in sequenza arbitraria
- Esistono anche connessioni asincrone (**queued**)
 - Codice dopo **emit** eseguito subito, poi gli slot

Meta-Object Compiler

- Il **moc** è un programma che gestisce le estensioni di Qt al C++
- Se una dichiarazione di classe contiene la macro **Q_OBJECT**, il **moc** produce altro codice C++ per quella classe
- Tra le altre cose, il "**meta-object code**" è necessario per il meccanismo di segnali e slot
- **Segnali e slot sono una estensione specifica di Qt alla sintassi C++**

Bottone con click destro

```
class RightPushButton : public QPushButton {  
    Q_OBJECT  
public:  
    using QPushButton::QPushButton;  
protected:  
    void mouseReleaseEvent(QMouseEvent* e);  
signals:  
    void rightClicked(); // new signal, in addition to QPushButton::clicked  
};
```

C++

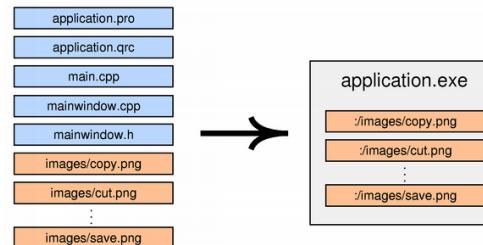
```
void RightPushButton::mouseReleaseEvent(QMouseEvent* e) {  
    if (e->button() == Qt::RightButton) emit rightClicked();  
    QPushButton::mouseReleaseEvent(e); // base class behaviour  
}
```

C++

Altre caratteristiche di Qt



Altre caratteristiche di Qt



- Possibile inserire dati (img, snd ecc.) direttamente nel file eseguibile
 - Aggiungere un "**Qt Resource File**" al progetto
 - Aggiungere le singole risorse al descrittore **.qrc**
 - Percorso delle risorse richiede ":" come prefisso

Gruppo di bottoni

- **QButtonGroup**: raggruppamento *logico* di bottoni
- Non fornisce **nessuna rappresentazione visuale**
- Utile per associare i bottoni ad un indice intero
 - Definisce il segnale **buttonClicked**, che trasmette come parametro l'indice del bottone
 - Possibile connettere questo segnale anziché il segnale **clicked** di ciascun bottone
- Utile anche per raggruppare bottoni radio
 - Gestisce lo stato di tutti i bottoni nel gruppo
 - Se **exclusive**, solo un bottone selezionato

Utilizzare le traduzioni

```
int main(int argc, char* argv[]) {
    QApplication a(argc, argv);

    // run lupdate(.pro->.ts), linguist and lrelease(.ts->.qm), first!
    QTranslator appTranslator;
    appTranslator.load(":translations/myapp_"
                      + QLocale::system().name());
    a.installTranslator(&appTranslator);

    QTranslator qtTranslator;
    qtTranslator.load("qt_" + QLocale::system().name(),
                     QLibraryInfo::location(QLibraryInfo::TranslationsPath));
    a.installTranslator(&qtTranslator);

    return a.exec();
}
```

QString

- Caratteri QChar a 16 bit (UTF-16, rari simboli a due QChar)
- Metodi static: `fromStdString`, `number`
- Metodi: `toStdString`, `toInt`, `toFloat`
- Metodo `split` (genera una `QStringList`)
- Sostituzione automatica di numeri, caratteri e stringhe
 - `QString status = QString{"Processing file %1 of %2: %3"}.arg(i).arg(total).arg(fileName);`
- Operatori [], >, <, ==, +, += ecc.



<Domande?>

Michele Tomaiuolo
Palazzina 1, int. 5708
Ingegneria dell'Informazione, UniPR