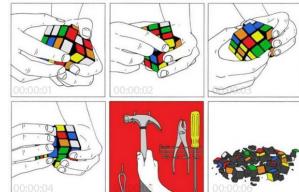




Algoritmi in Python 3



Introduzione alla programmazione

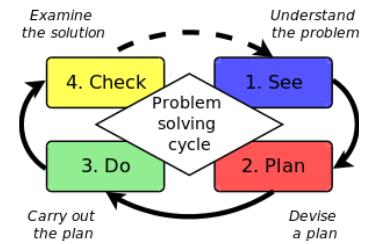
Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Problem solving

- George Polya (1945). *"How to solve it"*
 - Metodo per la soluzione di problemi matematici
 - Problem solving: raramente processo lineare

(1) See. Capire il problema

- Quali sono i dati, quali le incognite?
- Quali sono le condizioni?
- Sono soddisfacibili, ridondanti, contraddittorie?
- Figure, notazione



www.ce.unipr.it/people/tomamic

2/36

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

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

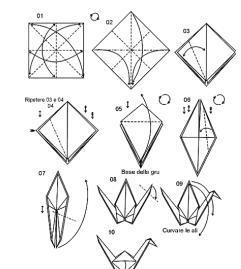
- Su schedario **ordinato** si può fare più in fretta
 - (1) **Finchè** restano delle schede da esaminare: si prende tra loro la scheda centrale
 - (2) **Se** autore e titolo sono quelli cercati: scheda trovata, ricerca conclusa!
 - (3) **Altrimenti, se** autore e titolo vengono dopo quelli cercati: 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

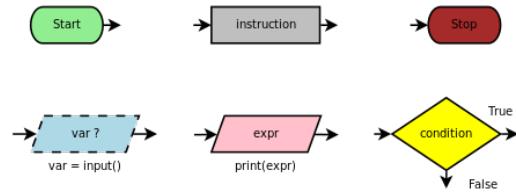
Diagramma di flusso

- 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

Python!



Tipi di dati



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

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

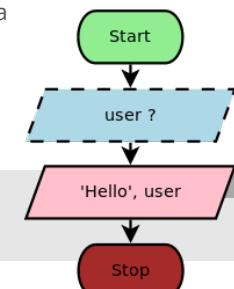
Tabella ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	(NULL)	32	20	[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	(EOT)	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	(LINE FEED)	42	2A	*	74	4A	K	106	6A	j
11	B	(VERTICAL TAB)	43	2B	+	75	4B	L	107	6B	k
12	C	(FORM FEED)	44	2C	,	76	4C	M	108	6C	l
13	D	(CARRIAGE RETURN)	45	2D	.	77	4D	N	109	6D	m
14	E	(SHIFT OUT)	46	2E	-	78	4E	O	110	6E	n
15	F	(SHIFT IN)	47	2F	/	79	4F	P	111	6F	o
16	10	(DEVICE CONTROL)	48	30	0	80	50	Q	112	70	p
17	11	(DEVICE CONTROL 1)	49	31	1	81	51	R	113	71	q
18	12	(DEVICE CONTROL 2)	50	32	2	82	52	S	114	72	r
19	13	(DEVICE CONTROL 3)	51	33	3	83	53	T	115	73	s
20	14	(DEVICE CONTROL 4)	52	34	4	84	54	U	116	74	t
21	15	(NEGATIVE ACKNOWLEDGE)	53	35	5	85	55	V	117	75	u
22	16	(SYNCHRONOUS IDLE)	54	36	6	86	56	W	118	76	v
23	17	(END OF TRANS. BLOCK)	55	37	7	87	57	X	119	77	w
24	18	(CANCEL)	56	38	8	88	58	Y	120	78	x
25	19	(END OF MEDIUM)	57	39	9	89	59	Z	121	79	y
26	1A	(SUBSTITUTE)	58	3A	:	90	5A	[122	7A	z
27	1B	(ESCAPE)	59	3B	,	91	5B	{	123	7B	{
28	1C	(FIELD SEPARATOR)	60	3C	<	92	5C	\	124	7C	\
29	1D	(GROUP SEPARATOR)	61	3D	=	93	5D	I	125	7D	j
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)
```



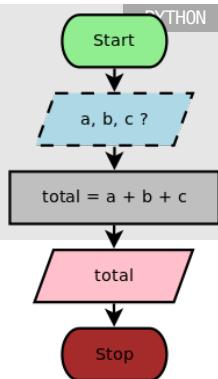
Somma di tre numeri

```
# 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

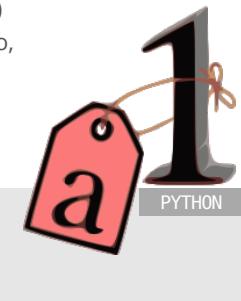
print("The sum is", total)
```



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!
```



Programmazione strutturata

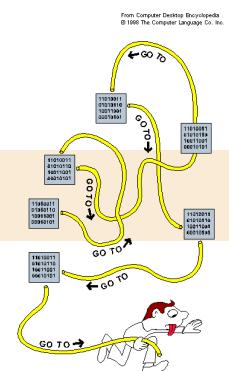


Programmazione strutturata

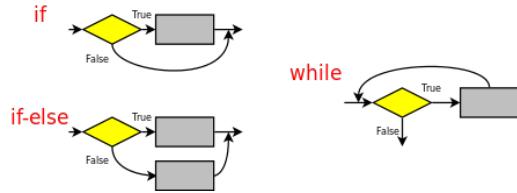
- Strutture di controllo:
 - **Sequenza**
 - **Selezione**
 - **Iterazione**

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

“*Goto statement considered harmful (Dijkstra, 1968)*”



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"

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 sse la condizione non è verificata

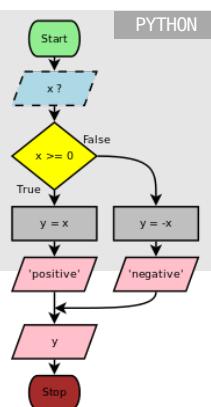
Valore assoluto

```
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)

PYTHON

Espressione booleana composta con **and** e **or**

Confronto tra parole

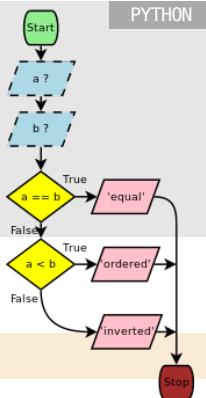
```
a = input("First word? ")
b = input("Second word? ")

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**

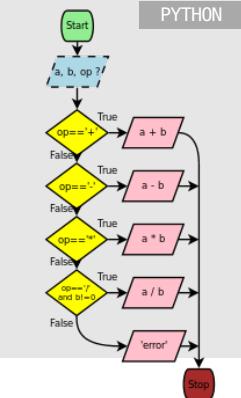
"There should be one -- and preferably only one -- obvious way to do it (ZoP)"



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")
```



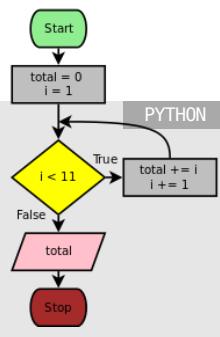
Iterazione: while

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

```
# Sum of the numbers from 1 to 10
total = 0
i = 1

while i < 11:
    total = total + i
    i = i + 1

print("The sum is", total)
# ... Yet Gauss' method is smarter
```



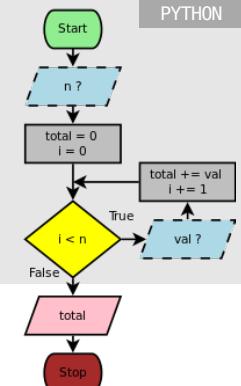
Somma di N valori

```
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)
```



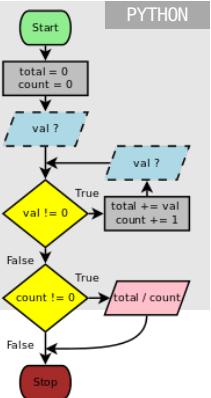
Media di sequenza di valori

```
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)
```

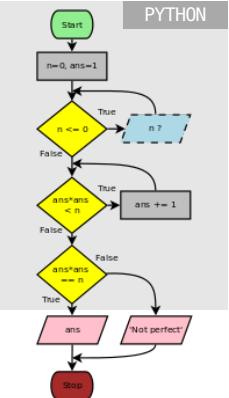


Quadrato perfetto

```
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")
```



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
```

3.141592653589793238462643383279
5028841971693993751058209749445923
07816068034327641573986280197242117067
9821 48086 5132
823 06647 09384
46 23350 59523
17 23359 4081
2848 1117
1052 4430
2701 9385
21105 55964
44929 44944
9303 81964
45653 134465
66593 34461
284756 49233
2019093 31642 71
9234603 48610454356468
2724587 0729241427
3724587 00660631558
817488 152092096

Esercizi

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 ...*



33/36

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



34/36

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
```



35/36

<Domande?>



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



Sequenze di dati

Introduzione alla programmazione

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR



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']
```

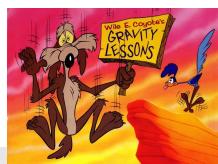
months	
"Jan"	months[0]
"Feb"	months[1]
"Mar"	months[2]
...	

www.ce.unipr.it/people/tomamic

2/28

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

PYTHON

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

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

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

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

PYTHON

www.ce.unipr.it/people/tomamic

3/28

www.ce.unipr.it/people/tomamic

4/28

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', 'milk']

print('Your shopping list contains:')

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

PYTHON

Ad ogni iterazione, a **product** è assegnato un diverso elemento della lista **shopping_list**

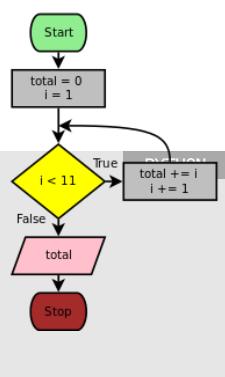
Intervalli di valori: range

- **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()
```

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

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

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)
```

PYTHON

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

PYTHON

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

PYTHON

Tupla

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

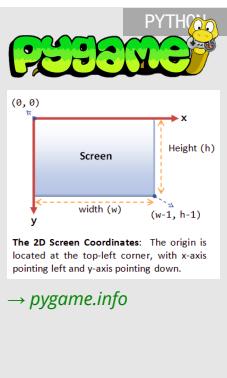
```
# Tuple packing  
pt = 5, 6, 'red'  
pt[0] # 5  
pt[1] # 6  
pt[2] # 'red'
```

PYTHON

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

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)  
  
# Blue circle, center=(300, 50), radius=20  
pygame.draw.circle(screen, (0, 0, 255), (300, 50), 20)  
  
# Yellow rectangle, left=50, top=75, w=90, h=50  
pygame.draw.rect(screen, (255, 255, 0), (50, 75, 90, 50))  
  
pygame.display.flip() # Update the screen  
while pygame.event.wait().type != pygame.QUIT:  
    pass  
pygame.quit()
```



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)



```
tel = {'john': 4098, 'terry': 4139}

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

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

PYTHON

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

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  
  
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)
```

PYTHON

```
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

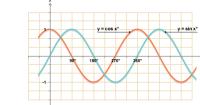
Usare un ciclo `for` sulla stringa (sequenza di caratteri)



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

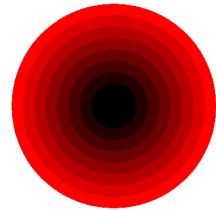
`sin, pi in modulo math`
`sin opera su radianti: rad = x * pi / 180`



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 ciclo, 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 `rows×cols`
 - 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*
- Per cominciare, inserire nella lista valori numerici crescenti, anziché lettere*
- Cella a inizio riga: il suo indice `i` è multiplo di `cols`, ossia `i % cols == 0`*
- Cella a fine riga: `i % cols == cols - 1`*



 www.ce.unipr.it/people/tomamic

26/28

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

 www.ce.unipr.it/people/tomamic

27/28



Funzioni

Introduzione alla programmazione

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR



Funzione

- Operatore, applicato a operandi, per ottenere un risultato
 - def** per definire una funzione
 - return** per terminare e restituire un risultato

```
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)
```

PYTHON

www.ce.unipr.it/people/tomamic

2/27

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.: modifiche *permanent*i
- Si possono restituire più valori, come *tupla*
 - return** 7, 5, 'black'

```
def inc(a):
    a += 1  # try to print a, now
x = 10
inc(x)  # try to print x, now
```

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

www.ce.unipr.it/people/tomamic

4/27

3/27

Effetti collaterali

- Modifica di oggetti passati come parametri o variabili globali, operazioni di lettura/scrittura...
- Annullo 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

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

PYTHON

Programmazione ricorsiva



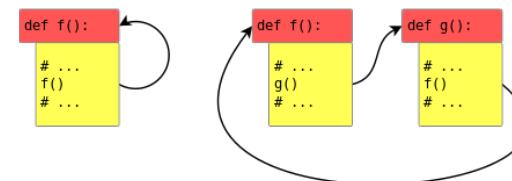
DII

Programmazione ricorsiva

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

Direct call

Indirect call

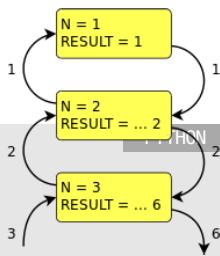


Fattoriale, ricorsione

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

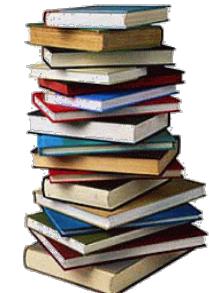
```
def factorial(n: int) -> int:
    result = 1
    if (n > 1):
        result = n * factorial(n - 1)
    return result
```

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

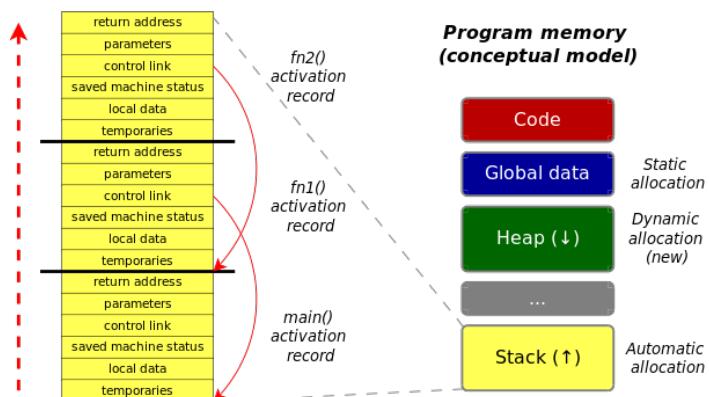


Stack dell'applicazione

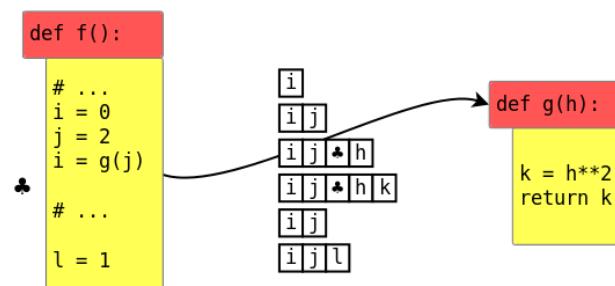
- Pila: memoria dinamica **LIFO (Last In First Out)**
 - Dimensione massima prefissata
- Il programma ci memorizza automaticamente:
 - Indirizzo di ritorno** per la funzione
Inserito alla chiamata, estratto all'uscita
 - Parametri** della funzione
Inseriti alla chiamata, eliminati all'uscita
 - Variabili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



Record di attivazione



Vista semplificata dello stack



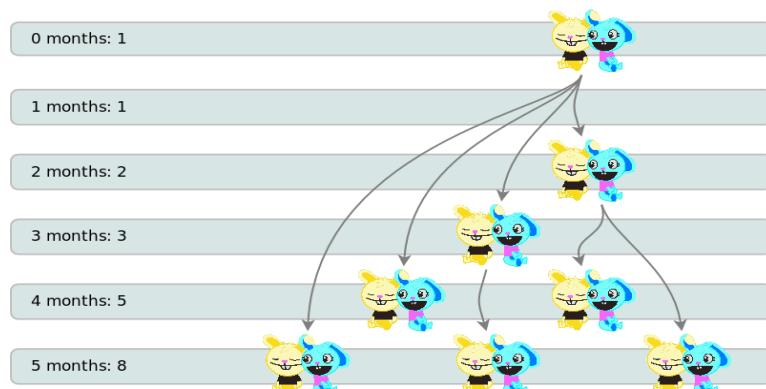
Visibilità di una variabile

- 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!



Esempi di ricorsione

I conigli di Fibonacci

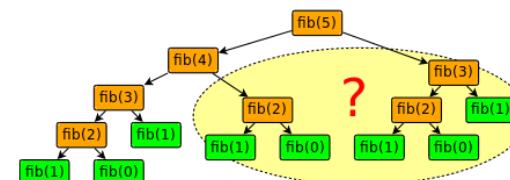


$fib(0) = fib(1) = 1; fib(n) = fib(n-1) + fib(n-2);$

Fibonacci, ricorsione

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

PYTHON



Fibonacci, memoization

```
_fibonacci_lookup = [1, 1]

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

# alternative: fibonacci._lookup = [1, 1]
# a function is an object and can have fields!
```

PYTHON

Fibonacci, iterazione

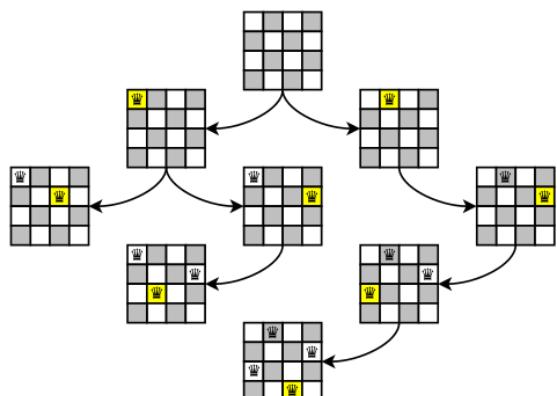
```
def fibonacci(n: int) -> int:
    result = 1
    fib2 = 1 # result @ 2 steps before

    for i in range(2, n + 1):
        result += fib2
        # store previous result, for next step
        fib2 = result - fib2

    return result;
```

PYTHON

N regine, backtracking



N regine, verifica

```
def under_attack(board, row, col) -> bool:
    # for each direction up-left, up, up-right...
    # (there are no queens in lower cells)
    dy = -1
    for dx in [-1, 0, +1]:

        # walk till finding a queen, or border
        x, y = col + dx, row + dy
        while 0 <= y < len(board) and 0 <= x < len(board[y]):

            # a queen is found -> square under attack
            if board[y][x]: return True

            x, y = x + dx, y + dy
    return False
```

PYTHON

N regine, ricorsione

```
def place_queens(board, row) -> bool:  
    for col in range(len(board[row])):  
        if not under_attack(board, row, col):  
            # square not attacked, place a queen  
            board[row][col] = True  
  
            # is this the last row?  
            if row == len(board) - 1: return True  
  
            # else, place queens in the following rows  
            if place_queens(board, row + 1): return True  
  
            # no luck this way, remove the queen  
            board[row][col] = False # (backtracking)  
    return False
```

PYTHON

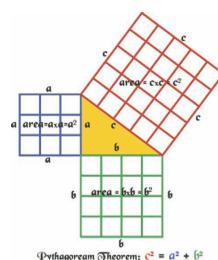


DII

Esercizi

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



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: $MCD(m, n) = MCD(n, m \% n)$



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

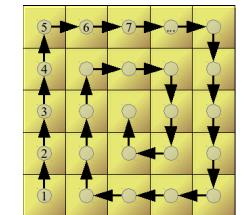
- Scrivere una funzione per riempire di numeri crescenti una matrice quadrata (o rettangolare)
- Seguire il percorso a spirale suggerito nella figura a fianco
- Dimensioni della matrice indicate dall'utente a runtime

Tenere traccia della direzione attuale (Δy , Δx)
Avanzare fino al bordo o ad una cella già visitata,
poi cambiare la direzione in senso orario

Rotazione +90°: $(x', y') = (y, -x)$

Rotazione -90°: $(x', y') = (-y, x)$

In generale: $(x', y') = (\cos(\theta) \cdot x + \sin(\theta) \cdot y, -\sin(\theta) \cdot x + \cos(\theta) \cdot y)$



<Domande?>

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



Introduzione alla
programmazione

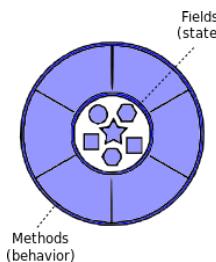
Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Oggetti e grafica



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

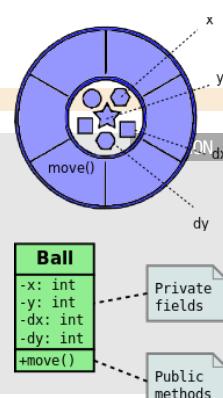


Definizione della classe

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

“Siamo tutti adulti consenzienti. (GvR)”

```
class Ball:  
    ARENA_WIDTH = 16  
    ARENA_HEIGHT = 12  
  
    def __init__(self, x: int, y: int):  
        self._x = x  
        self._y = y  
        self._dx = 1  
        self._dy = 1  
  
    # ...
```



Class diagram UML

Costruzione oggetti

- **__init__**: metodo **iniziatore**
 - 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(4, 8) # Allocation and initialization
```

PYTHON

Metodi

- Espongono **servizi** ad altri oggetti

```
class Ball:  
    # ...  
    def move(self):  
        new_x = self._x + self._dx  
        if not (0 <= new_x < Ball.ARENA_WIDTH):  
            self._dx = -self._dx  
        new_y = self._y + self._dy  
        if not (0 <= new_y < Ball.ARENA_HEIGHT):  
            self._dy = -self._dy  
        self._x += self._dx  
        self._y += self._dy  
  
    def position(self) -> (int, int):  
        return self._x, self._y
```

PYTHON

Applicazione

```
from ball import Ball # Ball is defined in ball.py  
import sys  
  
# Create two objects, instances of the Ball class  
b1 = Ball(4, 8)  
b2 = Ball(8, 4)  
print('Ball 1 @', b1.position())  
print('Ball 2 @', b2.position())  
  
for line in sys.stdin:  
    b1.move()  
    b2.move()  
    print('Ball 1 @', b1.position())  
    print('Ball 2 @', b2.position())
```

PYTHON

www.ce.unipr.it/people/tomamic

6/36

www.ce.unipr.it/people/tomamic

7/36

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

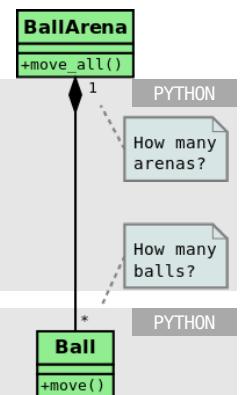
```
ball = Ball(4, 8)  
print('ball @', ball.pos)  
# ball.pos = (6, 2) # with the setter, you could change the pos
```

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()
```



www.ce.unipr.it/people/tomamic

8/36

www.ce.unipr.it/people/tomamic

9/36

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...
 - *Generational garbage collection*: per oggetti creati di recente
 - *Mark & sweep*: per oggetti più duraturi

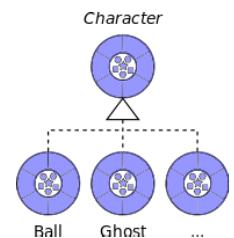


Livelli di astrazione

Livelli di astrazione

- Relazione **is-a** tra classi
 - Specializzazione, sotto-insieme
- **Character**: *classe base*
 - Dichiara un metodo **move** ecc.
- Vari attori: *classi derivate*
 - Ereditano caratteristiche di **Character**
 - Definiscono comportamenti specifici

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



PYTHON

Generalizzazione e riuso

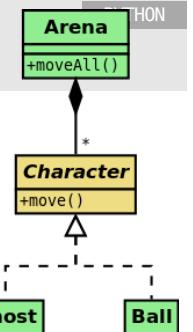
```
class Arena: # ...
    def __init__(self):
        self._characters = []
    def add(self, a: Character):
        self._characters.append(a)
    def move_all(self):
        for a in self._characters:
            a.move()
```

PYTHON

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

Sostituzione

```
arena.add(Ball(4, 8))
arena.add(Ghost(12, 4)) # ...
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 Character che contiene
- Relazione **is-a** tra classi derivate (Ball e Ghost) e classe base (Character)

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(Character): # ...
    def move(self):
        dx = random.choice([-1, 0, 1])
        dy = random.choice([-1, 0, 1])
        self._x = (self._x + dx) % self._w
        self._y = (self._y + dy) % self._h
```

PYTHON

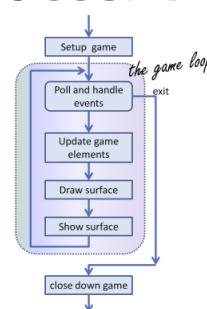


Grafica con Pygame

Grafica con pygame

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

pygame.org



Struttura di un gioco

PYTHON

```
import pygame
pygame.init() # Prepare pygame
clock = pygame.time.Clock() # To set game speed
screen = pygame.display.set_mode((640, 480)) # (w, h)
image = pygame.image.load('ball.bmp')

playing = True
while playing:
    for e in pygame.event.get(): # Handle events: mouse, keyb etc.
        if e.type == pygame.QUIT: playing = False
    # **Apply game logic here**
    screen.fill((25, 225, 75)) # BG (Red, Green Blue)
    screen.blit(image, (90, 60)) # FG (x, y)
    pygame.display.flip() # Surface ready, show it!
    clock.tick(30) # Delay to get 30 fps
pygame.quit() # Close the window
```

www.ce.unipr.it/people/tomamic

18/36

www.ce.unipr.it/people/tomamic

19/36

Rimbalzi: setup

```
arena = Arena(16, 12)
Ball(arena, 4, 8) # ...

TILE_SIDE = 20
SCREEN_SIZE = (arena.width * TILE_SIDE,
               arena.height * TILE_SIDE)
BACKGROUND = (255, 255, 255)

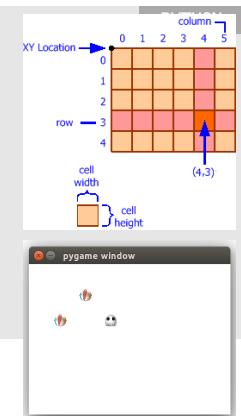
pygame.init()
clock = pygame.time.Clock()
screen = pygame.display.set_mode(SCREEN_SIZE)
images = {Ball.SYMBOL: pygame.image.load('ball.bmp'),
          Ghost.SYMBOL: pygame.image.load('ghost.bmp')}
# ...
```

PYTHON

Rimbalzi: ciclo

```
while playing: # ...
    arena.move_all() # Game logic

    screen.fill(BACKGROUND)
    for y in range(arena.height):
        for x in range(arena.width):
            symbol = arena.get_symbol(x, y)
            if symbol in images:
                i = images[symbol]
                p = (x * TILE_SIDE, y * TILE_SIDE)
                screen.blit(i, p)
    pygame.display.flip() # ...
```



www.ce.unipr.it/people/tomamic

20/36

www.ce.unipr.it/people/tomamic

21/36

Tastiera e mouse

```
from pygame.locals import (KEYDOWN, KEYUP, K_RIGHT,
    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

Disegni e suoni

```
# Blue circle, center=(300, 50), radius=20, width=0
# If width > 0, only outline (width is optional)
pygame.draw.circle(screen, (0, 0, 255), (300, 50), 20, 0)
# Yellow rectangle, left=50, top=75, w=90, h=50
pygame.draw.rect(screen, (255, 255, 0), (50, 75, 90, 50))
```

PYTHON

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

PYTHON

Testo

```
# 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



Gui con Tkinter (...)

Gui con Tkinter

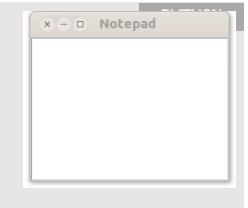
- Tk: libreria leggera e intuitiva, per interfacce grafiche
- In Python modulo `tkinter` (*Tk interface*)
- Portable tra diversi sistemi
- Usa le primitive grafiche della piattaforma ospite
 - → Efficiente anche su sistemi embedded
- Windows, MacOS, Linux, vari Unix...
- Tcl, Ruby, Python, Perl



Hello Notepad

```
from tkinter import Tk, Text

window = Tk()
text_edit = Text(window)
text_edit.pack(expand=1, fill='both')
window.title('Notepad')
# event loop: mouse, keyb etc.
window.mainloop()
```



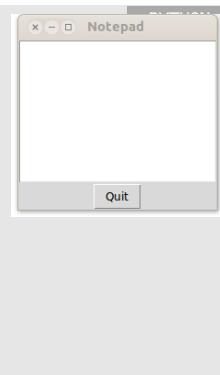
Aggiungere un bottone

```
from tkinter import Tk, Text, Button

window = Tk()
text_edit = Text(window)
text_edit.pack()

# When button is clicked, window's
# destroy method is executed
quit_button = Button(window, text='Quit',
                     command=window.destroy)
# widgets in vertical layout
quit_button.pack()

window.title('Notepad')
window.mainloop()
```



Classe per il Notepad

```
from tkinter import Tk, Text, Button, messagebox
PYTHON
class Notepad(Tk):
    def __init__(self):
        super().__init__()
        # basic widgets as private fields
        self._text = Text(self)
        self._text.pack()
        self._quit = Button(self, text='Quit', command=self.exit)
        self._quit.pack()
    def exit(self):
        if messagebox.askokcancel('Quit', 'Are you sure?'):
            self.destroy()
    if __name__ == '__main__':
        win = Notepad()
        win.mainloop()
```

Creazione di un menù

```
from tkinter import Tk, Text, Menu

class Notepad(tk.Tk):
    def __init__(self):
        super().__init__()
        text_edit = Text(self)
        text_edit.pack()

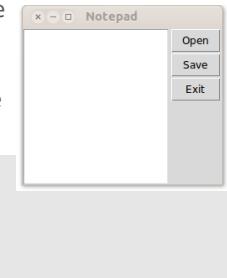
        menu_bar = Menu(self) # tearoff=0
        self.config(menu=menu_bar)
        menu_file = Menu(menu_bar)
        menu_file.add_command(label='Open')
        menu_file.add_command(label='Save')
        menu_file.add_command(label='Exit', command=self.exit)
        menu_bar.add_cascade(label='File', menu=menu_file) # ...
```

PYTHON

Disposizione widget

- Tk usa il meccanismo dei **geometry manager** per disporre i widget
- Ci sono tre sistemi di disposizione principali
- È possibile comporre più **Frame** (riquadri), per realizzare gerarchie di contenimento tra widget

```
# vertical layout, default side='top'
widget.pack()
# horizontal layout
widget.pack(side='left')
# grid layout
widget.grid(column=x, row=y)
```



Riquadri compositi

```
from tkinter import Tk, Text, Frame, Button

class Notepad(Tk):
    def __init__(self):
        super().__init__()
        text_edit = Text(self)
        text_edit.pack(side='left')

        v_box = Frame(self)
        v_box.pack(side='left') # fill='y'
        open_btn = Button(v_box, text='Open')
        open_btn.pack() # side='top'
        save_btn = Button(v_box, text='Save')
        save_btn.pack()
        exit_btn = Button(v_box, text='Exit')
        exit_btn.pack()
```

PYTHON

Layout a griglia

- Divide in una griglia lo spazio disponibile e dispone i widget nelle celle
- Specificare riga e colonna all'inserimento (l'indice parte da 0)
- Possibile specificare anche l'occupazione di più celle adiacenti

7	1	14	2
12		6	11
5	10	4	3
9	13	15	8

FifteenGui – Costruttore

```
class FifteenGui(Tk):
    def __init__(self, puzzle: FifteenPuzzle):
        super().__init__()
        self._puzzle = puzzle
        self._cols = puzzle.cols()
        self._rows = puzzle.rows()
        for y in range(self._rows):
            for x in range(self._cols):
                b = Button(self, command=lambda x=x, y=y:
                           self.handle_click(x, y))
                b.grid(column=x, row=y)
        self.update_buttons()
    # ...
```

PYTHON

FifteenGui – Click

```
class FifteenGui(Tk):
    # ...
    def handle_click(self, x: int, y: int):
        self._puzzle.move_position(x, y)
        self.update_buttons()
        if self._puzzle.is_finished():
            messagebox.showinfo('Congrats', 'Puzzle solved')
            self.destroy()

    def update_buttons(self):
        for y in range(self._rows):
            for x in range(self._cols):
                val = str(self._puzzle.get(x, y))
                # Get the button on the grid @ (x, y)
                b = self.grid_slaves(column=x, row=y)[0]
                b['text'] = val
```

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;
}
```

C++

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

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: **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";

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

C++

Operazioni di base

- Operazioni su numeri: `+`, `-`, `*`, `/`, `%`
 - Anche incremento e decremento unitario: `++`, `--`
 - **Attenzione**: la divisione tra interi dà risultato intero (`trunc`)
 - **Attenzione**: il resto della divisione può essere negativo
 - Assegnamento: `=`, `+=`, `-=` ...
 - Confronti: `>`, `>=`, `<`, `<=`, `!=`, `==`
- Operazioni booleane (and, or, not): `&&`, `||`, `!`
- Stringhe: operazioni di confronto, concatenazione: `+`
 - **Attenzione**: apici doppi per valori `string`, singoli per `char`

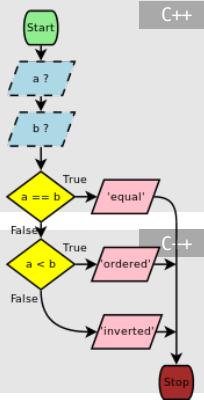
```
string txt = "Lorem ipsum";
txt[6] = 'I';
cout << txt[6]; // 'I'
```

C++

Decisioni

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

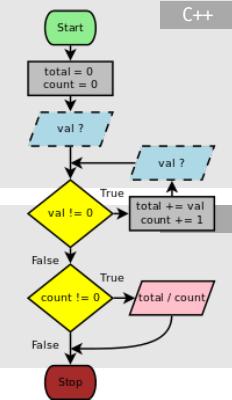
```
int choice; cin >> choice;
switch (choice) {
    case 1: cout << "First option"; break;
    case 2: cout << "Second option"; break;
    default: cout << "Error";
}
```



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);
```

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



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");
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, '-'));
```

C++

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;  
}
```

C++

Flussi e file

```
#include <fstream>  
// ...  
  
int n; float r; string w;  
ifstream in{"input.txt"}; // file input stream  
if (in.good()) { // is stream available?  
    in >> n >> r >> w;  
}  
in.close();  
  
ofstream out{"output.txt"}; // file output stream  
if (out.good()) { // is stream available?  
    out << "Values: " << n << " " << r << " " << w << endl;  
}  
out.close();
```

C++

Lettura di righe

```
ifstream in{"input.txt"}; // file input stream  
  
string first_line, second_line;  
getline(in, first_line);  
getline(in, second_line);  
// read lines do not include newline  
  
string whole_text;  
getline(in, whole_text, '\0');  
  
string line;  
while (getline(in, line)) {  
    cout << line << endl;  
}
```

C++

Ciclo di lettura

```
#include <iomanip>
// ...

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

C++

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

C++

Flussi e stringhe

- `istringstream`, `ostringstream` in libreria `sstream`

```
#include <sstream>
// ...

int n; float r; string w;
istringstream in{"5 7.5 hello"};
in >> n >> r >> w;
```

```
ostringstream out;
out << "Values: " << n << " " << r << " " << w << endl;
string text = out.str();
cout << text << endl;

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

C++

Oggetti

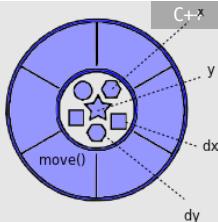


Oggetti

- 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*

Definizione: ball.h

```
class Ball {  
public:  
    Ball(int x0, int y0);  
    void move();  
    string str();  
    int get_x();  
    int get_y();  
  
    static const int ARENA_WIDTH = 16;  
    static const int ARENA_HEIGHT = 12;  
  
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 = 1; dy = 1;  
}  
  
void Ball::move() {  
    if (x + dx < 0 || x + dx >= ARENA_WIDTH) dx = -dx;  
    if (y + dy < 0 || y + dy >= ARENA_HEIGHT) dy = -dy;  
    x += dx; y += dy;  
}  
  
string Ball::str() {  
    return to_string(x) + ", " + to_string(y);  
}
```



www.ce.unipr.it/people/tomamic

C++

19/36

Applicazione: main.cpp

```
#include "ball.h"  
// ...  
int main() {  
    Ball ball1{4, 4};  
    Ball ball2{8, 4};  
  
    string line;  
    while (getline(cin, line)) {  
        ball1.move();  
        ball2.move();  
  
        cout << ball1.str() << endl;  
        cout << ball2.str() << endl << endl;  
    }  
}
```

Allocazione dinamica

```
// ...  
int main() {  
    Ball ball1{4, 4};  
    Ball* ball2 = new Ball{8, 4};  
    // 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.str() << endl;  
        cout << ball2->str() << endl << endl;  
    }  
    delete ball2;  
}
```



www.ce.unipr.it/people/tomamic

C++

21/36



20/36

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/python3.3/ -std=c++11
g++ -shared ball.o ball_wrap.o -o _ball.so
```

```
>>> from ball import Ball
>>> b = Ball(6, 6)
>>> b.move()
>>> print(b.str())
```

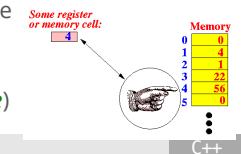
FILE: BALL.I

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*)



```
int i = 56;
int* p;           // a ptr to some int (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()`

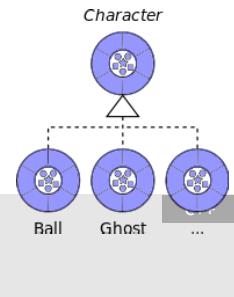
Livelli di astrazione



DII

Livelli di astrazione

- Character: *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 Character {
    virtual void move() = 0;
    // ...
};
```

Composizione

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

C++

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

C++

Ereditarietà e polimorfismo

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

C++

```
class Ghost: public Character {
// ...
Ghost() {
// ...
}
void move() {
    int dx = (rand() % 3) - 1;
    int dy = (rand() % 3) - 1;
    x = (x + dx) % w;
    y = (y + dy) % h;
}
};
```

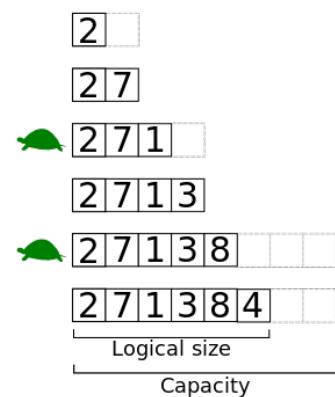
C++



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 interi

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

Implementazione nel repository di esempi

C++

Inserimento in vector

```
void IntVector::insert(int pos, int 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 IntVector::expand_capacity() {
    capacity_ *= 2;
    int* bigger = new int[capacity_];
    for (int i = 0; i < size_; i++) bigger[i] = data_[i];
    delete[] data_;
    data_ = bigger;
}
```

C++

www.ce.unipr.it/people/tomamic

30/36

www.ce.unipr.it/people/tomamic

31/36

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 interi

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

C++

Implementazione nel repository di esempi

www.ce.unipr.it/people/tomamic

32/36

www.ce.unipr.it/people/tomamic

33/36

Inserimento in lista

```
void IntList::insert(int pos, int 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. vector di float, string ecc.: cambia solo il tipo di dato!
- **Template**: meccanismo di programmazione generica in C++
 - Generazione trasparente di codice specializzato per tipi specifici
 - Es. `vector<int>` genera una classe simile a quella descritta prima

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

C++

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

C++

<Domande?>

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



Introduzione alla
programmazione

Michele Tomaiuolo
Ingegneria dell'Informazione, UniPR

Gui con Qt



35/36

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

2/57

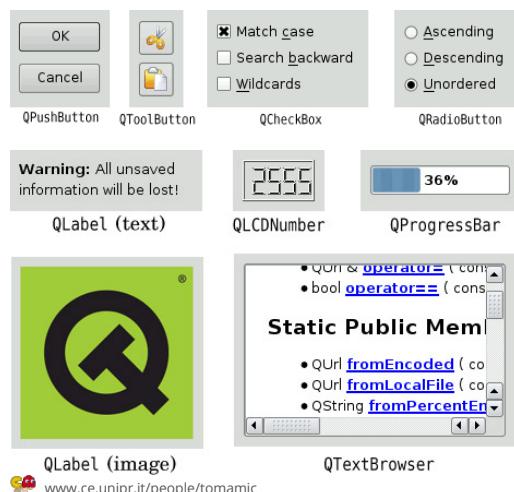
Libreria modulare

- 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

 www.ce.unipr.it/people/tomamic

3/57

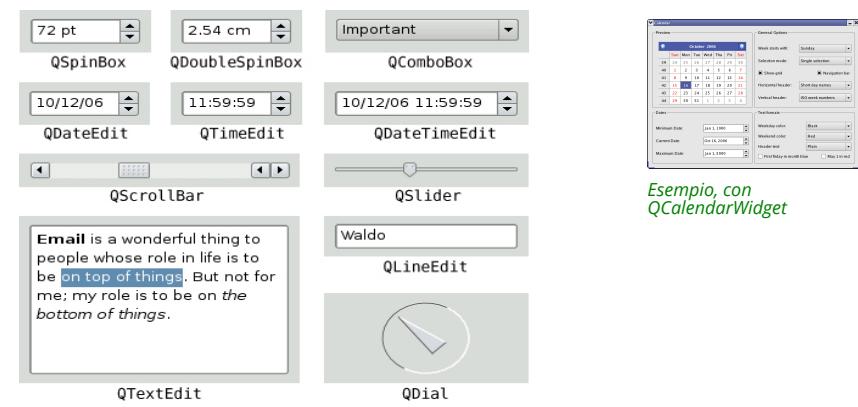
Bottoni e display



 www.ce.unipr.it/people/tomamic

4/57

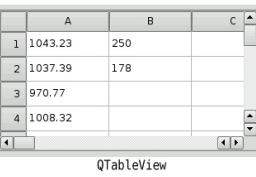
Input



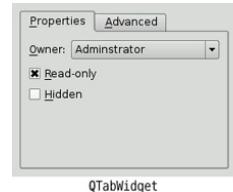
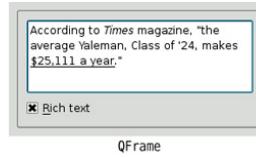
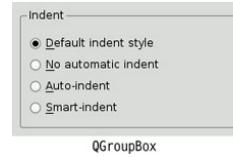
 www.ce.unipr.it/people/tomamic

5/57

Viste di elementi

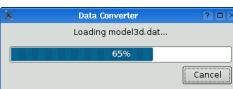


Contenitori

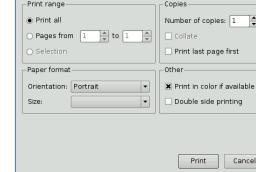
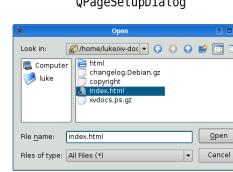
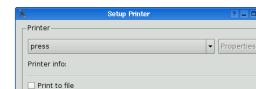
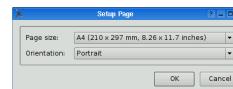


Inoltre: *QMenu, QMenuBar, QToolBar, QStatusBar*

Finestre di dialogo



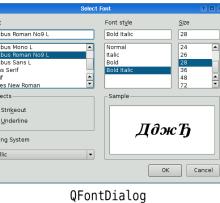
File e stampa



Colori e font



QColorDialog



QFontDialog

Stili

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

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("E&xit")};  
  
    // widgets in a vertical layout  
    setLayout(new QVBoxLayout); // QWidget  
    layout()->addWidget(text_edit);  
    layout()->addWidget(exit_button);  
}
```



Layout principali

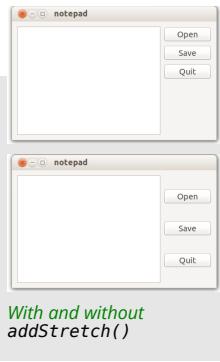
- Qt usa il meccanismo dei **layout** per disporre i widget
- Ci sono tre classi di layout principali
 - **QHBoxLayout**
 - **QVBoxLayout**
 - **QGridLayout**
- Per installare un layout su un widget, dobbiamo invocare il metodo **setLayout** del widget
- Il layout gestisce l'area **interna** al widget

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);
```



Click dei bottoni

Definire gli slot

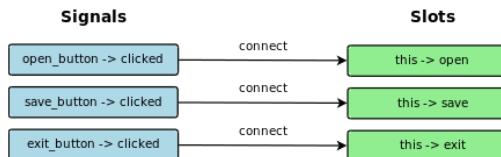
```
class Notepad : public QWidget {
    // add support for signals and slots...
    Q_OBJECT
public:
    Notepad();
public slots: // special methods, to connect with signals
    void open();
    void save();
    void exit();
private:
    QTextEdit* text_edit;
    QPushButton* open_button;
    QPushButton*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

C++

Connettere segnali e slot

```
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);
}
```

C++



Accoppiamento tra segnali e slot

Accoppiamento lasco

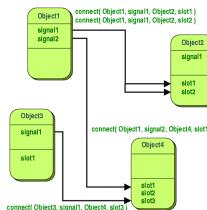
- Un oggetto emette un **segnale**, ma non sa quali **slot** lo ricevono
- Molti segnali ad un singolo slot, un segnale a molti slot

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

```
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

```
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"));  
        }  
    }  
}
```

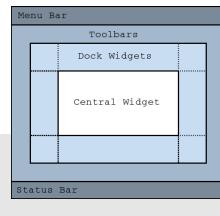
Chiudere l'app

```
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
```

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);
```



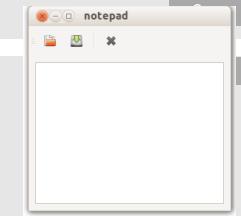
Menù e toolbar

```
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



Griglia di buttoni

- **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



FifteenGui

```
class FifteenGui : public QWidget {
    Q_OBJECT
public:
    FifteenGui(FifteenPuzzle* puzzle);
private:
    void handle_click(int x, int y);
    void update_button(int x, int y);
    void update_all_buttons();

    int cols() { return puzzle_->cols(); }
    int rows() { return puzzle_->rows(); }

    vector<QPushButton*> buttons_;
    FifteenPuzzle* puzzle_;
};
```



FifteenGui – Costruttore

```
auto grid = new QGridLayout; setLayout(grid);
for (auto y = 0; y < rows(); ++y) {
    for (auto x = 0; x < cols(); ++x) {
        auto b = new QPushButton;
        buttons_.push_back(b);
        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**

FifteenGui – Aggiornamento buttoni

```
void FifteenGui::update_button(int x, int y) {
    auto val = puzzle_->get({x, y});
    auto symbol = 'A' + val - FifteenPuzzle::FIRST;
    if (val == FifteenPuzzle::BLANK) symbol = ' ';

    auto b = buttons_[y * cols() + x];
    b->setText(QString{symbol});
}

void FifteenGui::update_all_buttons() {
    for (auto y = 0; y < rows(); y++)
        for (auto x = 0; x < cols(); x++)
            update_button(x, y);
}
```

C++

FifteenGui – Gestione click

```
void FifteenGui::handle_click(int x, int y) {
    puzzle_->move({x, y});
    auto blank = puzzle_->blank(), moved = puzzle_->moved();
    update_button(blank.real(), blank.imag());
    update_button(moved.real(), moved.imag());

    if (puzzle_->finished()) {
        QMessageBox::information(this, tr("Congratulations"),
                               tr("Game finished!"));
        puzzle_->shuffle();
        update_all_buttons();
    }
}
```



Pyside - Inizializzazione



Pyside

```
class FifteenGui(QWidget):
    def __init__(self, puzzle: FifteenPuzzle):
        super().__init__()
        self._puzzle = puzzle
        self._buttons = []
        self.setLayout(QGridLayout())
        for y in range(puzzle.rows):
            for x in range(puzzle.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()
    # ...
```



www.ce.unipr.it/people/tomamic

35/57

www.ce.unipr.it/people/tomamic

34/57

Pyside - Aggiornamento buttoni

```
class FifteenGui(QWidget):
    # ...
    def update_button(self, x: int, y: int):
        val = self._puzzle.get(x, y)
        symbol = chr(ord('A') + val - 1)
        if val == 0: symbol = ' '
        b = self._buttons[y * self._puzzle.cols + x]
        b.setText(symbol)

    def update_all_buttons(self):
        for y in range(self._puzzle.rows):
            for x in range(self._puzzle.cols):
                self.update_button(x, y)
```



Pyside - Gestione click

```
class FifteenGui(QWidget):
    # ...
    def handle_click(self, x: int, y: int):
        self._puzzle.move_pos(x, y)
        self.update_button(*self._puzzle.blank) # args unpacking
        self.update_button(*self._puzzle.moved)

        if self._puzzle.finished:
            QMessageBox.information(self, self.tr('Congratulations'),
                                    self.tr('Game finished!'))
            self._puzzle.shuffle()
            self.update_all_buttons()
```



www.ce.unipr.it/people/tomamic

37/57

www.ce.unipr.it/people/tomamic

36/57

Dispatching degli eventi

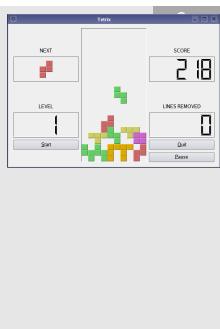


Eventi in Qt

- 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

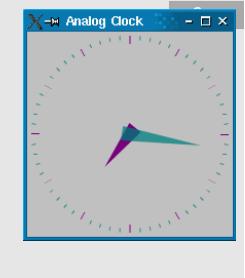
```
void TetrixBoard::keyPressEvent(QKeyEvent* e) {  
    switch (e->key()) {  
    case Qt::Key_Left:  
        tryMove(curPiece, curX - 1, curY);  
        break;  
        // ...  
    case Qt::Key_Down:  
        tryMove(curPiece.rotatedRight(), curX, curY);  
        break;  
        default:  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



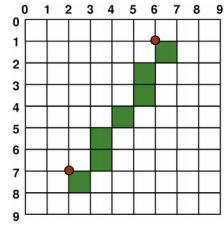
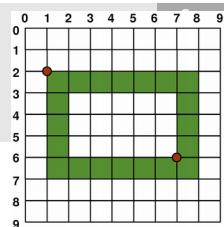
Disegni e animazioni

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);
```



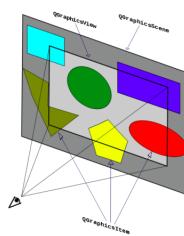
Disegno di immagini

- **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
 - Altrimenti, sfondo: **QWidget::setStyleSheet**
- **QPixmap** sono anche superfici di disegno custom
 - **QPainter::QPainter{QPaintDevice* d};**

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

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

Dichiarazione di segnali

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

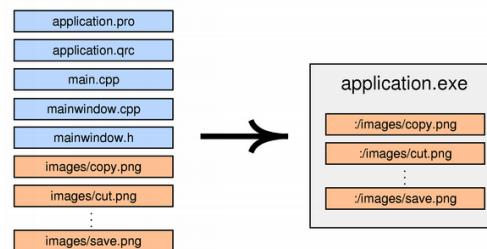
```
class RightPushButton : public QPushButton {  
    Q_OBJECT  
public:  
    using QPushButton::QPushButton;  
protected:  
    void mouseReleaseEvent(QMouseEvent* e);  
signals:  
    void rightClicked(); // new signal, in addition to QPushButton::clicked  
};
```

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

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 a 16 bit (UTF-16, rari simboli a due QChar)
 - `QChar.isHighSurrogate / isLowSurrogate`
- Metodi `fromStdString` e `toStdString`
- Metodi `setNum` e `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