Laboratorio d'Informatica

Corso di Laurea Magistrale in Ingegneria per l'Ambiente e il Territorio

A.A. 2021/2022

Docente: Lorenzo Putzu

Lezione 09

Il linguaggio Python

Struttura dei programmi: update

- Abbiamo già visto in precedenza i vantaggi della programmazione modulare, che, in particolare al crescere della complessità dei programmi da sviluppare, rende più semplici le operazioni di modifica e di testing del codice prodotto.

- La **programmazione ad oggetti** rappresenta un ulteriore sviluppo rispetto alla programmazione modulare. Il programma può essere visto come un insieme di oggetti che interagiscono tra loro.

- In questo tipo di programmazione si fa uso di un nuovo tipo di dato, la **classe**, che serve appunto a modellare un insieme di oggetti

Struttura dei programmi: update

- Una classe è uno strumento che ci permette di raggruppare variabili e funzioni nei nostri programmi in maniera logica e **riutilizzabile** il che ci consente di gestire progetti anche di grosse dimensioni in maniera molto semplice.

- Python è un linguaggio multi-paradigma, che supporta sia la programmazione procedurale (che è quella che abbiamo visto finora e che fa uso delle funzioni) sia la programmazione ad oggetti.

- Mentre nella programmazione procedurale le funzioni (o procedure) sono l'elemento organizzativo principale, nella programmazione ad oggetti l'elemento organizzativo principale sono gli **oggetti**.

Struttura dei programmi: update

- Nella **programmazione procedurale**, i dati e le funzioni sono separate, e questo può creare una serie di problemi, tra cui:
  - è necessario gestire dati e funzioni separatamente;
  - è necessario importare le funzioni che vogliamo usare;
  - è necessario passare i dati alle funzioni;
  - è necessario verificare che i dati e le funzioni siano compatibili;
  - è più difficile estendere e modificare le funzionalità;
  - il codice è più difficile da mantenere;
  - è più facile introdurre bug.

Struttura dei programmi: update

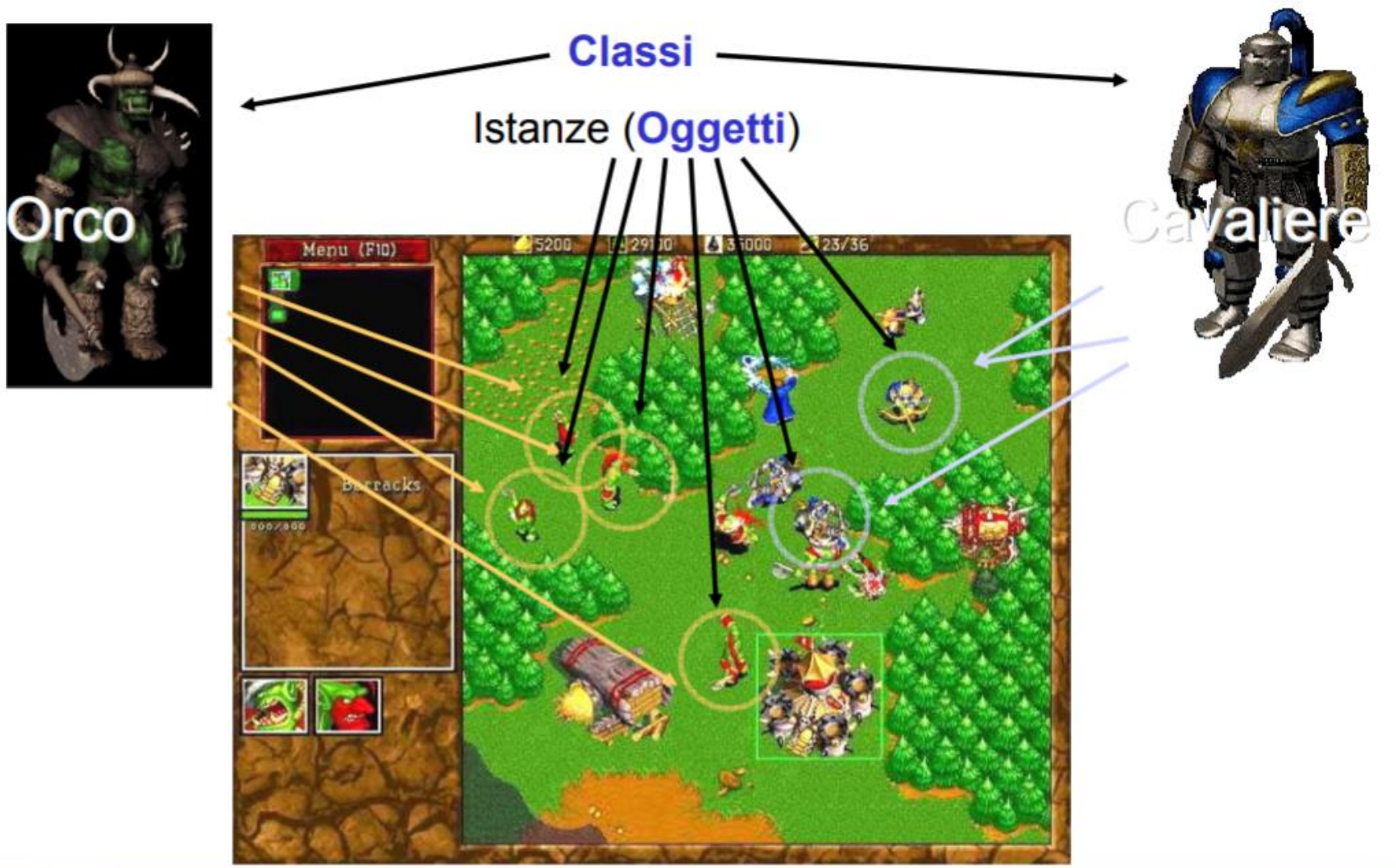
- Nella **programmazione ad oggetti**, gli oggetti svolgono la funzione di racchiudere in un'unica unità organizzativa sia i dati che e funzioni. Questo ha diversi vantaggi:
  - dati e funzioni sono raggruppati;
  - è facile sapere quali operazioni possono essere eseguite sui dati;
  - non è necessario importare funzioni per eseguire queste operazioni;
  - non è necessario passare i dati alle funzioni;
  - le funzioni sono compatibili con i dati;
  - è più facile estendere e modificare le funzionalità;
  - il codice è più semplice da mantenere;
  - è più difficile introdurre bug.

Classi e oggetti

- Le classi quindi permettono di modellare la realtà in maniera efficiente e riutilizzabile, infatti possiamo definire le proprietà di ciascuno oggetto in un **modello generico**, chiamato classe e da qui far derivare ciascun singolo individuo assegnandogli i suoi attributi personali.
- Ciascun oggetto creato a partire da questo modello generico viene chiamato **Istanza**.

- Esempio la fabbrica di biscotti:
  - lo stampo per i biscotti è il modello
  - ogni biscotto è un'istanza di quel modello
- Però per avere un'idea più precisa sulle potenzialità di una classe bisogna immaginare uno stampo che può dar vita a biscotti con caratteristiche differenti, assegnando differenti valori delle sue variabili di istanza o **attributi**.

Classi e oggetti:esempio



Classi e oggetti

Le **classi** quindi sono usate per definire le caratteristiche di un oggetto, i suoi attributi e i suoi metodi ma non si riferiscono a nessun oggetto specifico, in quanto rappresentano un modello che può essere usato per creare istanze.

Le **istanze** sono oggetti creati a partire da una classe. Quindi un'istanza di una classe si riferisce a uno specifico oggetto. Una classe può essere usata per creare diverse istanze dello stesso tipo ma con attributi diversi.

Definizione di classe

- Le classi introducono un pò di nuova sintassi e alcuni nuovi costrutti. La definizione di una nuova classe viene effettuata tramite la keyword **class** con la seguente sintassi

```
class nome_classe :
        sequenza di istruzioni
```

  dove nome_classe indica il nome che vogliamo dare alla classe
- La sintassi è molto simile a quella usata per la definizione di funzioni manca solamente la lista di argomenti in input. Mentre la sequenza di istruzioni può includere:
  - la definizione di nuovi metodi
  - la definizione di un metodo speciale chiamato _init_
  - la definizione di variabili di istanza
  - la definizione di variabili di classe

Instanziare una classe

- Una volta che abbiamo definito in maniera opportuna una nuova classe possiamo creare degli oggetti e quindi instanziare la classe con la seguente sintassi:
- **Sintassi**

```
nome_instanza = nome_classe(valore1, ...)
```

  - dove nome_instanza indica il nome che vogliamo dare all'oggetto che stiamo creando e invece nome_classe fa riferimento al nome che era stato assegnato alla classe in precedenza
  - valore1 invece indica un possibile argomento di input che verrà gestito all'interno della classe
- Il numero di argomenti di input dipende dalla definizione della classe è può anche essere vuoto

I metodi

- I **metodi** descrivono il comportamente dell'oggetto, sono simili alle funzioni ma sono specifici per ogni classe

```
class nome_classe :
        ...
        def nome_metodo(self):
        ...
```

- Come si può notare:
  - la loro definizione si trova indentata all'interno della classe
  - la sintassi usata per definire i metodi è uguale a quella usata per definire le funzioni
  - devono includere un parametro aggiuntivo che per convezione è chiamato **self**

I metodi

- **self** è un argomento che si riferisce all'istanza, e anche se i metodi devono dichiararlo esplicitamente, non è necessario passarlo esplicitamente, infatti immaginando di aver instanziato la classe precedente con

```
nome_instanza = nome_classe(...)
```

- possiamo invocare il metodo semplicemente come

```
nome_instanza.nome_metodo(...)
```

- Python in realtà risale in automatico alla classe di appartenenza di nome_instanza ed esegue

```
nome_classe.nome_metodo(nome_instanza,...)
```

- Per questo motivo ogni metodo definito nella classe deve presentare come primo argomento self

I metodi

- Le classi supportano anche diversi metodi “speciali” che sono identificati dalla presenza di due underscore prima e dopo il nome. Questi metodi non vengono chiamati direttamente con nome_instanza.__nome_metodo__, ma vengono in genere chiamati automaticamente in situazioni particolari
- Uno di questi metodi speciali è il **costruttore** definito come **__init__** chiamato automaticamente ogni volta che un’istanza viene creata:

```
class nome_classe :
        def __init__(self,...):
        …
```

- Gli argomenti passati durante la creazione dell’istanza vengono ricevuti da __init__. Questo ci permette di creare automaticamente istanze diverse in base agli argomenti passati in quanto in genere vengono utilizzati per definire quelli che sono gli attributo o variabili di un’istanza

Attributi

- Gli attributi di **istanza** sono i più comuni e si dichiarano all'interno di un metodo (in particolare l'init)
- Gli attributi di **classe** invece sono meno comuni in quanto sono legati alla classe, e perciò ogni instanza di tale classe avrà in comune tali attributi

```
class nome_classe :
        self.attributo = espressione
        def __init__(self,valore,...):
            self.attributo = valore
            ...
        ...
```

Esempio

- Definiamo una classe che rappresenta un rettangolo generico

```
class Rettangolo:
    def __init__(self, base, altezza):
        self.base = base
        self.altezza = altezza

    def calcola_area (self):
          return self.base * self.altezza

    def calcola_perimetro (self):
        return (self.base + self.altezza)*2
```

- Possiamo creare un'instanza della classe Rettangolo con l'istruzione

```
rettangolo1 = Rettangolo(10,20)
```

Esempio

- Una volta create l'instanza possiamo accedere a tutti i suoi attribute d'istanza che sono *base* e *altezza*

```
rettangolo1.base
rettangolo1.altezza
```

- E possiamo richiamarne i metodi *calcola_area* e *calcola_perimetro*

```
rettangolo1.calcola_area()
rettangolo1.calcola_perimetro()
```

- Da notare che, a differenza di una funzione generica, non dobbiamo passare nessun parametro ai metodi, che invece utilizzando gli attributi dell'istanza

Caratteristiche degli oggetti

- Le istanze sono **mutabili** perciò possiamo modificarne lo stato agendo direttamente su uno dei suoi attribute e assegnandogli un nuovo valore, come ad esempio

```
rettangolo1.base = 30
```

- Possiamo passare un'istanza come parametro ad una funzione nel solito modo. Ad esempio per una stampa dei suoi attributi

```
def StampaRettangolo(rettangolo):
  print('base', rettangolo.base)
  print('altezza',rettangolo.altezza)
```

- Allo stesso tempo possiamo anche restituire un'istanza come valore di ritorno di una funzione, ad esempio

```
def RaddoppiaRettangolo(rettangolo):
  rettangolo.base *= 2
  rettangolo.altezza *= 2
  return rettangolo
```

Uguaglianza tra istanze

- Anche se due istanze hanno gli stessi attributi, nel nostro esempio hanno la stessa base e altezza, non è detto che siano uguali.
- Se assegniamo ad una variabile il rettangolo che avevamo creato in precedenza allora si tratterà di un alias dello stesso oggetto perciò l'uguaglianza sarà verificata.
- Altrimenti dati due rettangoli, anche se hanno le stesse coordinate non fanno riferimento allo stesso oggetto ma a due oggetti diversi. Per verificare che due oggetti abbiamo lo stesso contenuto è necessario fare un confronto tra il loro attributi

```
def ConfrontaRettangoli(ret1, ret2):
  return (ret1.base==ret2.base) and (ret1.altezza==ret2.altezza)
```

Esercizi

1. Definire una classe di nome `Persona` le cui istanze contengano il nome, il cognome e la data di nascita (giorno, mese e anno, rappresentati come numeri interi) di una persona. Tali valori devono essere specificati come argomenti del costruttore della classe (ovvero del metodo `__init__`). Definire inoltre i metodi per accedere a tali valori; la data di nascita dovrà essere restituita dal metodo corrispondente sotto forma di stringa, nel formato `gg/mm/aaaa`
2. Definire una classe di nome `Conto_Corrente` per memorizzare informazioni sui conti correnti di una certa banca. Ogni istanza deve contenere il codice IBAN di un conto (27 caratteri alfanumerici), il nome, il cognome e il codice fiscale del titolare, il saldo e lo scoperto (il massimo saldo passivo ammesso). Tutti questi valori dovranno essere specificati come argomenti del costruttore della classe; i valori di *default* (predefiniti) di saldo e scoperto devono essere entrambi zero. Definire inoltre i metodi per accedere a ciascuno di tali valori, per assegnare un nuovo scoperto, per modificare il saldo dopo un versamento, e per modificarlo dopo un prelievo; quest'ultimo metodo dovrà consentire solo prelievi che non portino il saldo oltre lo scoperto, e dovrà restituire il valore dell'importo prelevato, se consentito, oppure 0

Esercizi

3. Definire una classe di nome `Studente` le cui istanze contengano il nome, il cognome, il codice fiscale, il numero di matricola (cinque cifre) e la data d'immatricolazione (sotto forma di dizionario con chiavi `'g'`, `'m'` e `'a'`) di un certo studente a un certo corso di studio universitario, e tutti gli esami da lui superati (codice dell'insegnamento – cinque cifre –, data dell'esame – una stringa nel formato `gg/mm/aaaa` – e voto – un intero tra 18 e 30, oppure 31 per codificare ``30 con lode''). Tutti questi valori, eccetto le informazioni sugli esami, dovranno essere specificati come argomenti del costruttore della classe. Le informazioni sugli esami dovranno essere memorizzate come una lista di dizionari (un dizionario per esame), il cui valore iniziale dovrà essere una lista vuota. Definire i metodi per accedere alle informazioni su ciascuno studente: nome, cognome, codice fiscale, numero di matricola, data d'immatricolazione (da restituire sotto forma di stringa nel formato `gg/mm/aaaa`) ed elenco degli esami (che dovrà essere stampato sullo schermo); definire inoltre un metodo per registrare un nuovo esame, avente per argomenti il codice, la data (una stringa, si veda sopra) e il voto (con la stessa codifica numerica indicata sopra)

Esercizi

4. Definire una classe per rappresentare un mazzo di 52 carte da gioco, composte dai quattro semi Cuori, Quadri, Fiori, Picche, e dai valori 2, 3, ..., 10, J, Q, K, A. Ogni carta dovrà essere rappresentata da un dizionario con due chiavi (seme e valore, i cui valori dovranno essere stringhe), e il mazzo dovrà essere rappresentato come una lista di dizionari. Il mazzo iniziale dovrà contenere le 52 carte ordinate per seme (C, Q, F, P) e, all'interno di ciascun seme, per valori crescenti (2, ..., A). Definire quindi i metodi che eseguano le seguenti operazioni:
   - stampare le carte del mazzo nel formato (seme,valore), (seme,valore), ...
   - mescolare il mazzo (cioè disporre le carte in un ordine casuale)
   - "tagliare" ("alzare") il mazzo rispetto alla posizione centrale
   - estrarre e restituire (sotto forma di dizionario) la prima carta (rimuovendola dal mazzo)
   - estrarre e restituire (sotto forma di dizionario) una carta qualsiasi (rimuovendola dal mazzo)

Suggerimento: usare le funzioni della libreria `random` (`shuffle`, `randint`, ...)