Il linguaggio Python

Laboratorio d'Informatica

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

A.A. 2019/2020

Docente: Giorgio Fumera

Python (v. 2): richiami

Assegnamento, espressioni, tipi di dato

- Istruzione di assegnamento
- Tipi di dato elementari:
  - numeri interi
  - numeri frazionari
  - stringhe
- Espressioni:
  - **aritmetiche**: operatori `+`, `-`, `*`, `/`, `//` (quoziente intero), `**` (potenza), `%` (modulo)
  - **stringhe**: l'operatore di concatenazione `+`

Istruzioni composte

- Istruzione condizionale
  - `if...else...`
  - `if...`
  - `if...elif...else...`
  - `if...elif...`

- Istruzione iterativa: `while`

Ingresso/uscita (input/output)

- L'istruzione `print`

- Funzioni per l'acquisizione di valori attraverso la tastiera
  - `input`
  - `raw_input`

Esercizi

1. Calcolare le radici di un'equazione di secondo grado per valori dati dei coefficienti (se le radici non sono reali, stampare un messaggio opportuno): $ax^2 + bc + c = 0$
2. Un anno è bisestile se è multiplo di 100 e divisibile per 400, oppure se **non** è multiplo di 100 ma è divisibile per 4. Questa regola vale **solo** per gli anni successivi al 1582. Scrivere un programma che acquisisca un anno, verifichi se è successivo al 1582, e in questo caso stabilisca se sia bisestile o meno
3. Scrivere un programma che acquisisca una sequenza di numeri di lunghezza data e ne calcoli la somma
4. Scrivere un programma che acquisica una sequenza di numeri fino all'inserimento del valore 0, e ne calcoli la somma
5. Modificare il programma precedente in modo che calcoli anche il numero di valori positivi inseriti

Exercises

1. Compute the roots of a given second-degree equation (if the roots are complex, print a suitable message): $ax^2 + bc + c = 0$
2. A year is bissextile if it is a multiple of 100 and is divisibile by 400, or if it is **not** a multiple of 100 and it is divisibile by 4. This rule is valid only for the years after 1582. Write a program that reads a year and checks if it comes after 1582; if so, the program must determine if it is bissextile or not
3. Write a program that reads a sequence of numbers of a given length and computes their sum
4. Write a program that reads a sequence of numbers up to the occurrence of the first zero, and computes their sum
5. Modify the previous program so that it also computes the number of positive values in the sequence

Esercizi

6. Calcolare il fattoriale di un dato numero non negativo n:

$$n! = 1 \times 2 \times \cdots \times (n-1) \times n, \text{ se } n > 0$$
$$n! = 1, \text{ se } n = 0$$

7. Calcolare la somma dei primi n termini della serie armonica per un dato intero non negativo n:

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

8. Per un dato $x \geq 0$ trovare il più piccolo intero n tale che:

$$\sum_{k=1}^{n} \frac{1}{k} > x$$

Exercises

6. Compute the factorial of a given non-negative integer n:

$$n! = 1 \times 2 \times \cdots \times (n-1) \times n, \text{ if } n > 0$$
$$n! = 1, \text{ if } n = 0$$

7. Compute the sum of the first n terms of the harmonic series for a given non-negative integer n:

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

8. For a given $x \geq 0$ find the smallest integer n such that:

$$\sum_{k=1}^{n} \frac{1}{k} > x$$

Definizione di nuove funzioni

Le istruzioni `def` **e** `return`**.**

Esercizi

9. Definire una funzione che riceva come argomento un numero intero non negativo n e ne restituisca il fattoriale

10. Definire una funzione che riceva come argomento un numero intero non negativo n e restituisca la somma dei primi n termini della serie armonica

Exercises

9. Define a function that takes as argument a non-negative integer n and returns its factorial

10. Define a function that takes as argument a non-negative integer n and returns the sum of the first n terms of the harmonic series

Liste

- Rappresentazione
- Accesso agli elementi e loro modifica: indicizzazione
- Operatori: concatenazione, confronto
- Funzioni predefinite (`len`, `min`, `max`, `sum`, `range`, ...)
- Costruzione per concatenazione
- L'istruzione iterativa `for` per la scansione di liste

Liste e stringhe: *sequenze*

- Operatori: indicizzazione, concatenazione, *slicing*, `in`, `not in`
- La funzione predefinita `len`
- L'istruzione `for` per la scansione di sequenze
- Sequenze **mutabili** (liste) e **immutabili** (stringhe)

Esercizi

11. Definire una funzione (analoga a `range`) che, dato un intero n, restituisca una lista composta da tutti gli interi da 0 a $n-1$, se n è positivo, oppure una lista vuota
12. Definire una funzione (analoga a `sum`) che, data una lista composta da numeri, resituisca la loro somma. Definire due versioni della funzione: usando `while` e usando `for`
13. Definire una funzione che, date due liste composte da numeri e aventi la stessa lunghezza, restituisca una lista della stessa lunghezza contenente le somme dei loro elementi. Esempio: [4, 1, 2], [0, -3, 7] → [4, -2, 9]

Exercises

11. Define a function analogous to `range` that, given an integer n, returns a list containing all the integers between 0 and $n - 1$, if $n > 0$, and an empty list otherwise
12. Define a function analogous to `sum` that, given a list containing numbers, returns their sum. Define two versions of this function: one using `while` and another using `for`
13. Define a function that, given two lists of identical length containing numbers, returns their element-by-element sum stored in a list of the same length. For instance: [4, 1, 2], [0, -3, 7] → [4, -2, 9]

Esercizi (iterazioni nidificate)

14. Stampare la tavola pitagorica dei numeri da 1 a 10 sotto forma di una matrice:

```
1  2    3    ...  9    10
2  4    6    ...  18   20
...              ...        ...
1020   30   ...  90   100
```

15. Definire una funzione che acquisisca come argomento una lista L e restituisca una nuova lista contenente tutti gli elementi che compaiono in L **una sola volta**. Esempio: [3, 1, 5, 1, 6, 2, 9, 6] → [3, 5, 2, 9]

Exercises (nested iterations)

14. Print the multiplication table of numbers from 1 to 10, as a matrix:

```
1 2    3    ...  9    10
2 4    6    ...  18   20
...              ...       ...
10     20   30   ...  90   100
```

15. Define a function that, given a list L, returns a new list containing all the elements that occur only once in L.
    An example: [3, 1, 5, 1, 6, 2, 9, 6] → [3, 5, 2, 9]

Dizionari

- Rappresentazione
- Accesso agli elementi e loro modifica
- Aggiunta di elementi
- Operatori di confronto
- Costruzione

Esercizi

16. Definire una funzione che riceva come argomenti tre numeri rappresentanti una data (giorno, mese e anno) e restituisca un dizionario contenente i tre valori associati alle chiavi `"giorno"`, `"mese"` e `"anno"`.
    Esempio: `{"giorno":8, "mese":10, "anno":2019}`

17. Definire una funzione che, dato un numero di persone, acquisisca nome, cognome ed età di ciascuna di esse e memorizzi tali dati in una lista di dizionari aventi chiavi `"nome"`, `"cognome"`, `"eta'"`.
    Esempio:
    `{"nome":"Ugo", "cognome":"Gui", "eta'":25}`

Exercises

16. Define a function that takes as arguments three integers representing a date and returns a dictionary containing such values associated to the keys `"day"`, `"month"` and `"year"`.
    An example: `{"day":8, "month":10, "year":2019}`

17. Define a function that, given a number of people, reads the name, surname and age of each of them and returns these values as a list of dictionaries, each one having keys `"name"`, `"surname"` and `"age"`. An example:
    `{"name":"John", "surname":"Smith", "age":25}`

Accesso ai *file*

- Le funzioni `open` e `close`
- Scrittura su *file*: la funzione `write`
- Lettura da *file*: le funzioni `read`, `readline`, `readlines` (le funzioni ausiliarie `int`, `float`, `str` e `split`)

Esercizi

18. Definire una funzione che riceva una lista di dizionari contenenti ciascuno nome, cognome ed età di una persona (come nell'es. 17), e scriva tali dati in un nuovo *file* di nome `dati_anagrafici.txt` (una riga per persona). Per esempio, i dati del dizionario:
    `{"nome":"Ugo", "cognome":"Gui", "eta'":25}`
    dovranno corrispondere alla riga:
    `Ugo Gui 25`

19. Definire una funzione senza argomenti che acquisisca i dati dal *file* creato nell'esercizio precedente, li memorizzi in una lista di dizionari aventi la stessa struttura indicata sopra, e restituisca tale lista

Exercises

18. Define a function that takes as argument a list of dictionaries having the same structure as in exercise 17, and writes such data into a text file named `data.txt` (one row per person). For instance, the data in the following dictionary:
    `{"name":"John", "surname":"Smith", "age":25}`
    should correspond to the following row in the file:
    `John Smith 25`

19. Define a function with no arguments that reads the data from the file above, stores them into a list of dictionaries having the same structure above, and returns that list

Esercizi di riepilogo

20. Definire una funzione che, data una lista L, restituisca una nuova lista ottenuta facendo "scorrere" di una posizione verso destra gli elementi di L e spostando l'ultimo elemento nella prima posizione.
    Esempio: `[3, -2, 4, 20]` → `[20, 3, -2, 4]`
21. Definire una funzione che, data una lista L, restituisca una nuova lista contenente gli elementi di L in ordine inverso.
    Esempio: `[3, -2, 4, 20]` → `[20, 4, -2, 3]`
22. Definire una funzione che, data una lista di dizionari come il seguente, contenenti i dati sugli studenti che hanno sostenuto un certo esame:
    `{"matricola":"12345", "cognome":"Gui", "nome":"Ugo", "voto":27},`
    stampi nella *shell* il numero e la media aritmetica dei voti degli studenti che hanno superato l'esame, e la matricola e il cognome degli studenti che hanno conseguito il voto 30 con lode (rappresentato dal valore 31).

Esercizi di riepilogo: campionato di basket

23. Un *file* di testo di nome `partite.txt` contiene i risultati di tutte le partite giocate nel corso di un campionato di pallacanestro. Ogni riga corrisponde a una partita, per esempio:

```
Milano Venezia 80 75
Trento Sassari 82 95
```

- scrivere una funzione senza argomenti che restituisca un dizionario contenente la classifica del campionato (2 punti per una vittoria, 0 per una sconfitta), e avente per chiavi i nomi delle squadre; per esempio:
  `{"Milano":20, "Trento":24, ...}`
- Scrivere una funzione che, dato un dizionario come quello precedente, stampi sulla *shell* la classifica in ordine decrescente di punteggio; esempio:

```
Trento 24
Milano 20
...
```

Esercizi di riepilogo: campionato di basket

Suggerimento:

- memorizzazione della classifica in un dizionario:
  - acquisire il contenuto del *file* con `readlines`
  - scandire le partite (le righe del *file*) e costruire un dizionario avente per chiavi i nomi delle squadre e zero (punti) come valore associato a ciascuna chiave; esempio: `{"Milano":0, "Trento":0, ...}`
  - scandire una seconda volta le partite e aggiungere nel dizionario due punti alla squadra vincente di ogni partita
- stampa della classifica:
  - costruire due liste contenenti una le chiavi (nomi delle squade), l'altra i valori (i punti delle stesse squadre) del dizionario, usando le funzioni (*metodi*) `keys` e `values`, che restituiscono chiavi e valori di un dizionario **nello stesso ordine**
  - ordinare le due liste, per esempio con l'algoritmo di ordinamento per selezione

Esercizi di riepilogo: il gioco del Tris

24. Scrivere un programma che consenta a due persone di giocare a Tris. Il programma deve chiedere a turno a ciascun giocatore le coordinate della casella nella quale desidera tracciare il proprio simbolo, mostrare la nuova configurazione della griglia, riconoscere se il gioco è terminato e dichiarare il vincitore o un pareggio.

    Si suggerisce di strutturare il programma in un insieme di funzioni, che dovranno essere poi richiamate dal programma principale (che potrà a sua volta essere esso stesso una funzione). Un esempio è riportato di seguito.

Esercizi di riepilogo: il gioco del Tris

- La griglia può essere stampata nella *shell* come segue, evidenziando gli indici delle righe e delle colonne:

```
  123
1|OX |
2| X |
3| O |
```

- La griglia può essere memorizzata come una lista composta a sua volta da tre liste (righe) di tre elementi ciascuna, corrispondenti ai caratteri "O", "X" e " " (uno spazio). Per esempio, la configurazione qui sopra sarà rappresentata come segue:
  `[["O", "X", " "], [" ", "X", " "], [" ", "O", " "]]`
- Definire una funzione che riceva come argomento una configurazione della griglia e la stampi sulla *shell*
- Definire una funzione che riceva come argomento una configurazione della griglia e verifichi a quale situazione di gioco corrisponde, restituendo un valore opportuno: vittoria di "O", vittoria di "X", pareggio, partita non conclusa
- Definire una funzione che riceva come argomento una configurazione della griglia, acquisisca tramite la tastiera le coordinate della casella nella quale uno dei giocatori desidera tracciare il proprio simbolo (verificando che la casella sia libera), e restituisca la nuova configurazione della griglia
- Scrivere il programma principale, che dovrà chiamare le funzioni indicate sopra. Tale programma può essere a sua volta scritto sotto forma di funzione

Concluding exercises

20. Define a function thath given a list L, returns a new list obtained by "shifting" by one position on the right the elements of L and moving the last element of L in the first position.
    An example: `[3, -2, 4, 20]` → `[20, 3, -2, 4]`
21. Define a function that, given a list L, returns a new list containing the elements of L in reverse order. An example: `[3, -2, 4, 20]` → `[20, 4, -2, 3]`
22. Define a function that takes as argument a list of dictionaries containing the outcome of an exam taken by a set of students, like the following one:
    `{"matriculation":"12345", "name":"John", "surname":"Smith", "grade":27}`
    The function should print on the shell the number of students who passed the exam (i.e., whose grade is higher than 17) and the corresponding average grade, as well as the matriculation number and the surname of the students who got full marks (coded as 31).

Concluding exercises: basketball championship

23. A text file named `matches.txt` contains the scores of all the matches played in a given basketball championship. Each row corresponds to one match, like in the following example:

```
Milano Venezia 80 75
Trento Sassari 82 95
```

- define a fuction with no arguments that reads the file and returns a dictionary containing the championship table (2 points for a win, 0 for a defeat), with the team names as keys; for instance: `{"Milano":20, "Trento":24, ...}`
- Define a fuction that, given a dictionary like the one above, prints on the shell the championship table, where the teams are sorted from the top to the bottom of the table; for instance:

```
Trento 24
Milano 20
...
```

Concluding exercises: basketball championship

Hints:

- creating the dictionary with the championship table:
  - read the file content using `readlines`
  - scan the matches (file rows) and build a dictionary with the team names as keys and zero (points) as the corresponding value; an example: `{"Milano":0, "Trento":0, ...}`
  - scan again the matches, and for each match add two points to the dictionary key corresponding to the winning team
- printing the championship table:
  - create two lists containing the keys (team names) and the values (their points) of the dictionary, using the functions (*methods*) `keys` and `values`, which return the keys and the values of a dictionary **in the same order**
  - sort the two lists, for instance using the selection sort algorithm

Concluding exercises: tic-tac-toe

24. Write a program that allows two users to play tic-tac-toe. The program should ask in turn to each player the coordinates of the square where he would like to place his own symbol, show the corresponding grid configuration, detect when the game is ended, and announce the winner (if any) or a draw.

    Hint: struture your program as a set of functions that are called by the main program (which can be a function itself), as in the following example.

Concluding exercises: tic-tac-toe

- The grid could be shown on the shell as follows, highlighting the row and column indexes:

```
  123
1|OX |
2| X |
3| O |
```

- The grid can be stored in memory as a list made up of lists of three elements corresponding to the characters "O", "X" and " " (a space). For instance, the above configuration can be represented as:
`[["O", "X", " "], [" ", "X", " "], [" ", "O", " "]]`
- Define a function that takes as argument a given grid configuration and prints it on the shell
- Define a function that takes as argument a given grid configuration and checks which game situation it corresponds to, returning a suitable value: "O" wins, "X" wins, draw, game to be continued
- Define a function that takes as argument a given grid configuration, reads the coordinates of the cell where one of the players would like to place his own symbol (checking whether that cell is free), and returns the corresponding grid configuration
- Write the main program, which has to call the above functions. The main program itself can be defined as function

Python 2: nuovi elementi

Tipi di dato predefiniti
Variabili come riferimenti a *oggetti*
Metodi (funzioni) per i tipi di dato predefiniti
Programmazione orientata agli oggetti

Tipi di dato predefiniti

- I tipi di dato **tuple** ("tupla") e **set** (insieme)
  - funzioni e operatori sulle tuple: `len`, `in`, `not in`, indicizzazione, *slicing*
  - funzioni e operatori sugli insiemi: `len`, `in`, `not in`
- Inizializzazione di stringhe, liste, tuple, insiemi e dizionari, e conversione di valori: le funzioni `str`, `list`, `tuple`, `set` e `dict`
- **Tipo** di un valore Python: la funzione `type`
- L'istruzione iterativa `for` per stringhe, liste, tuple, dizionari e insiemi

Metodi (funzioni) per i tipi di dato predefiniti

Principali **metodi** (funzioni) per l'elaborazione di **oggetti** (valori) dei tipi predefiniti:

- stringhe (operazioni non distruttive): `upper`, `lower`, `replace`, `count`, `find`
- liste (operazioni distruttive): `append`, `insert`, `index`, `pop`, `remove`, `sort`; `count`
- dizionari: `pop`, `keys`, `values`
- insiemi: `add`, `discard`, `remove`, `clear`, `issubset`, `union`, `intersection`, `difference`

Esercizi

25. Definire una funzione che riceva come argomento una stringa contenente una frase (che si assume non contenere caratteri di punteggiatura) e stampi nella *shell* tutte le parole presenti in essa in ordine crescente della loro lunghezza
26. Scrivere una funzione che restituisca una lista contenente tutti i numeri primi compresi tra 2 e un dato intero n ricevuto come argomento, usando il metodo detto *crivello di Eratostene*: partendo dalla sequenza di tutti i valori tra 2 e n, si eliminano prima i multipli di 2, poi i multipli del valore successivo (tra quelli rimanenti), e così via
27. Scrivere una funzione che riceva come argomento una lista di numeri e restituisca il *secondo* valore maggiore presente in essa.
    Esempi: [3, 6, 1, 5] → 5, [4, -2, 7, 5, 7, 1] → 7

Esercizi

28. Definire una funzione che riceva come argomento una lista e determini se in essa siano presenti valori duplicati (cioè valori che occorrono più di una volta), restituendo `True` oppure `False`
29. Un vettore è detto *sparso* se contiene pochi elementi diversi da 0. I vettori sparsi possono essere memorizzati in dizionari contenenti solo i valori degli elementi diversi da 0, associati a una chiave corrispondente alla loro posizione nel vettore. Per esempio, il vettore corrispondente alla lista [0, 0, -1, 0, 0, 0, 3, 0] può essere codificato dal dizionario {2:-1, 6:3}
    - scrivere una funzione che riceva una **lista** contenente un vettore sparso e restituisca il dizionario corrispondente
    - scrivere una funzione che riceva due **dizionari** rappresentanti vettori sparsi e restituisca un dizionario corrispondente alla loro somma (vettoriale)

Exercises

25. Define a function that takes as argument a string containing a sentence (which is assumed not to contain punctuation marks) and prints on the shell all the words in that sentence, sorted for increasing length
26. Define a function that takes as argument an integer n and returns a list containing all the prime numbers between 2 and n , using the following procedure: starting from a sequence of all integers between 2 and n, first remove all the multiples of 2, then all the multiples of the next value among the remaining ones, and so on
27. Define a function that takes as argument a list of numbers and returns its *second* largest value. Two examples: [3, 6, 1, 5] → 5, [4, -2, 7, 5, 7, 1] → 7

Exercises

28. Define a function that takes as argument a list and returns `True` if it contains duplicate values (that is, values occurring more than once), and `False` otherwise
29. A vector is called *sparse* if it contains a few elements different from 0. Sparse vectors can be stored into dictionaries containing only the values of their non-zero elements, using as keys the positions of such elements. For instance, the vector corresponding to the list [0, 0, -1, 0, 0, 0, 3, 0] can be represented by the dictionary {2:-1, 6:3}
    - define a function that takes as argument a **list** containing a sparse vector and returns the corresponding dictionary
    - define a function that takes as arguments two **dictionaries** corresponding to sparse vectors and returns another dictionary corresponding to their vector sum

Variabili come riferimenti a *oggetti*

- Variabili come **riferimenti** a **oggetti** (valori)
- Oggetti **mutabili** (liste, dizionari e insiemi) e **immutabili** (stringhe, tuple), e loro uso nelle funzioni

Programmazione orientata agli oggetti

- I concetti di *classe* e *oggetto* nei linguaggi orientati agli oggetti
  - **istanze** di una classe
  - valori: **variabili d'istanza**
  - operazioni eseguibili sulle istanze: **metodi** (funzioni)
- Definizione di una nuova classe
  - l'istruzione `class`
  - il metodo `__init__`
  - il parametro `self`
  - definizione di nuovi metodi
- Creazione di istanze di una classe

Esercizi

30) 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`
31) 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

32) 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

33) 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`, ...)

Exercises

30) Define a class named `Person` whose instances contain the name, surname and birth date (day, month and year, represented as integers) of an individual. These values should be provided as argument to the class constructor (that is, the method `__init__`). Define also the methods to access these values. The birth date should be returned by the corresponding method as a string, using the format `gg/mm/aaaa`
31) Define a class named `Bank Account` to store information on the accounts of a certain bank. Each instance should contain the IBAN code of an account (27 alphanumeric characters), the name, surname and tax code (a string) of the account holder, the credit balance and the maximum overdraft (balance due) allowed by the bank, as a negative value. The initial values of all instance variables should be provided as arguments to the class constructor; the default values of the credit balance and of the maximum overdraft should be both zero. Define the methods to access all such values, to change the maximum overdraft, to update the balance after a deposit, and to update it after a withdrawal; the latter method should allow only withdrawals that do not exceed the maximum overdraft, and should return the amount of the withdrawal, if allowed, and 0 otherwise

Exercises

32) Define a class named `Student` whose instances contain the name, surname, tax code, matriculation number (five digits) and matriculation date (a dictionary with keys `'d'`, `'m'` and `'y'`) of a student enrolled on a course of study, and all the examinations passed (course code – five digits –, date – a string in the format `dd/mm/yyyy` – and mark – an integer between 18 and 30, or 31 to denote ``full marks''). All such values, except for the exams information, should be given as arguments to the class constructor. The information on the exams should be stored as a list of dictionaries (one dictionary per exam), which should be initially empty. Define the methods to access the information about each student: name, surname, tax code, matriculation number, matriculation date (to be returned as a string in the format `dd/mm/yyyy`) and list of the exams (which should printed on the screen); define a further method to record a new examination, whose arguments should be the course code, the date (as a string, see above) and the mark (using the same numerical coding as above)

Exercises

33) Define a class to represent a deck of 52 cards, made up of the four suits Heart, Diamond, Club, Spade, and of the values 2, 3, ..., 10, J, Q, K, A. Each card should be represented as a dictionary with two keys (suit and value, both coded as strings), and the deck should be represented as a list of such dictionaries. The initial deck should contain the 52 cards sorted by suit (H, D, C, S) and, for each suit, sorted by increasing value (2, ..., A). Define methods to carry out the following operations:
    - print the cards in the current deck in the format (suit,value), (suit,value), ...
    - shuffle the cards (i.e., arranging the cards in a random order)
    - cut the cards in the middle
    - pick and return (as a dictionary) the first card of the deck (removing it from the deck)
    - pick and return (as a dictionary) a random card (removing it from the deck)

Hint: use the functions of the library `random` (`shuffle`, `randint`, ...)