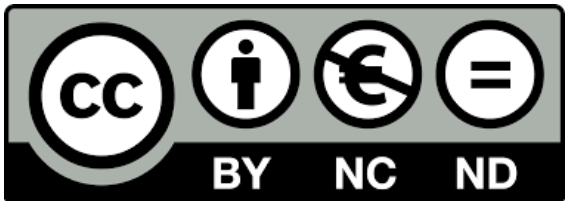
Introduzione alla programmazione

ITS - Umbria

A.S. 2022-23

MATERIA: fondamenti di informatica

Docente: prof. Paolo Bernardi



Dizionari: cosa sono

I dizionari di Python sono vettori (array) associativi, oggetti che in altri linguaggi di programmazione vengono normalmente indicati con "hash" o "mappings". Somigliano alle liste, ma gli indici (keys o chiavi) sono esclusivamente oggetti immutabili come le stringhe, i numeri o le tuple.

Internamente sono implementati con l'uso di "hash tables", per cui le chiavi devono essere oggetti "hashables", cioè oggetti che Python deve poter trasformare internamente in indirizzi; tutti gli oggetti immutabili hanno queste caratteristiche, ed anche oggetti istanze di classi create dall'utente, soddisfano questo criterio.

Si chiamano dizionari perché l'uso di stringhe per ritrovare elementi di un insieme e' analogo a quello che si fa quando si cerca una parola in un dizionario. Si usano per trattare insiemi di elementi contraddistinti da un nome, piuttosto che da un indice numerico, come per le liste.

Dizionari: caratteristiche

I dizionari sono oggetti mutabili e un dizionario può contenere oggetti eterogenei: numeri, stringhe, liste, dizionari.

Si possono costruire strutture complesse con dizionari e liste annidate.

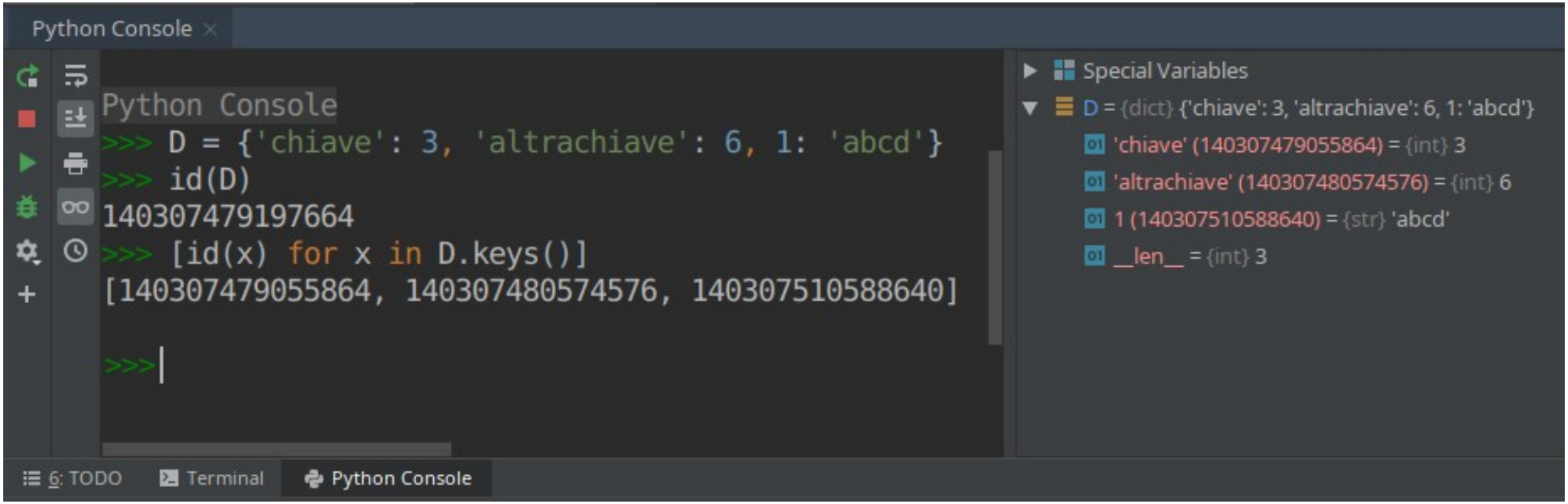
Gli elementi di un dizionario non hanno un ordine definito, e per individuare gli elementi si utilizza, al solito, una coppia di parentesi quadre, che contengono la chiave dell'elemento.

I dizionari non implementano gli operatori = e * come le liste. Per aggiungere un elemento basta definirne la chiave ed il valore.

Un dizionario è rappresentato da coppie di chiavi e valori, separati da virgole, e racchiusi fra parentesi graffe.

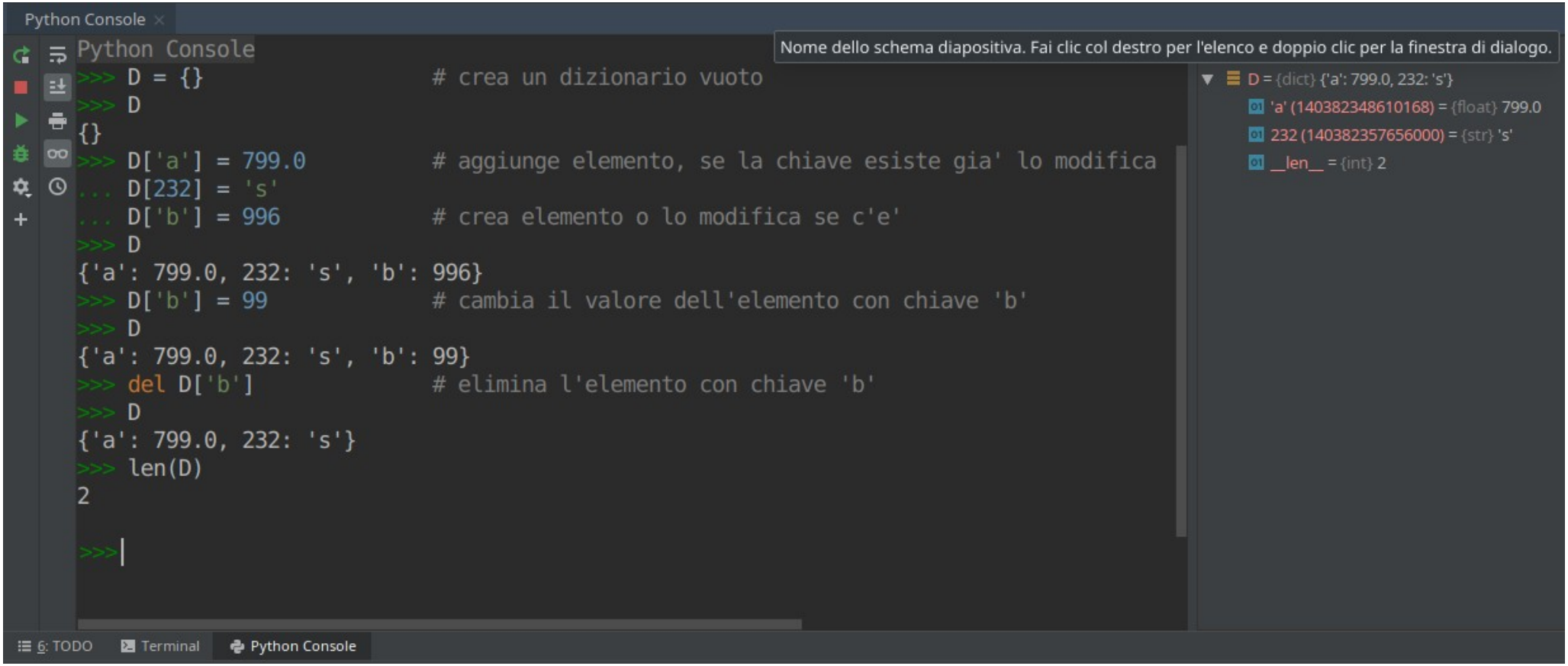
Dizionari: caratteristiche

Esempio



Il linguaggio Python (ver. 3)

Dizionari: caratteristiche



Dizionari: caratteristiche

```
Python Console
>>> D = {'a': 799.0, 232: 's', 'b': 996, 'c': 99}
>>> D
{'a': 799.0, 232: 's', 'b': 996, 'c': 99}
>>> D.pop('c')                  # estrae dal dizionario il valore con chiave 'c' e lo elimina.
99
>>> D
{'a': 799.0, 232: 's', 'b': 996}
>>> D.update({ 1: 10, 2: 20, 3:30 })    # unisce dizionari
>>> D
{'a': 799.0, 232: 's', 'b': 996, 1: 10, 2: 20, 3: 30}
>>> D.clear()                   # svuota il dizionario
>>> D
{}

>>>
```

6: TODO Terminal Python Console

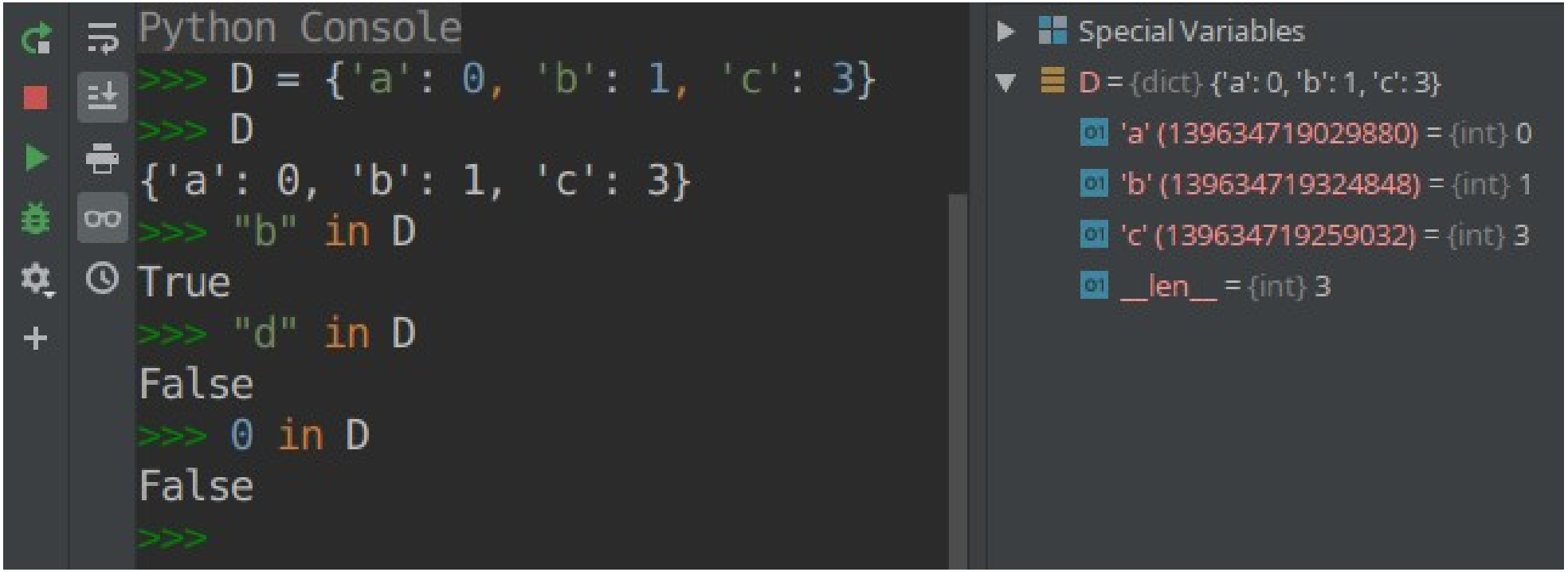
Dizionari: metodi

Per utilità si riporta l'elenco dei metodi dei dizionari:

Metodo	Descrizione
D.clear( )	Rimuove tutti gli elementi del dizionario
D.copy( )	Restituisce una copia superficiale (shallow copy) del dizionario
D.fromkeys(seq[, value]))	Crea un nuovo dizionario con le chiavi da seq e i valori impostati sul valore.
D.get(key, value)	Restituisce D[key] se la chiave esiste, altrimenti value
D.items( )	Restituisce un oggetto iterabile di coppie di tuple di D (chiave, valore)
D.keys( )	Restituisce un oggetto iterabile delle chiavi del dizionario
D.values( )	Restituisce un oggetto iterabile dei valori del dizionario

Dizionari: caratteristiche

La funzione "in" e' riferita esclusivamente alle chiavi, ad esempio



Dizionari: caratteristiche

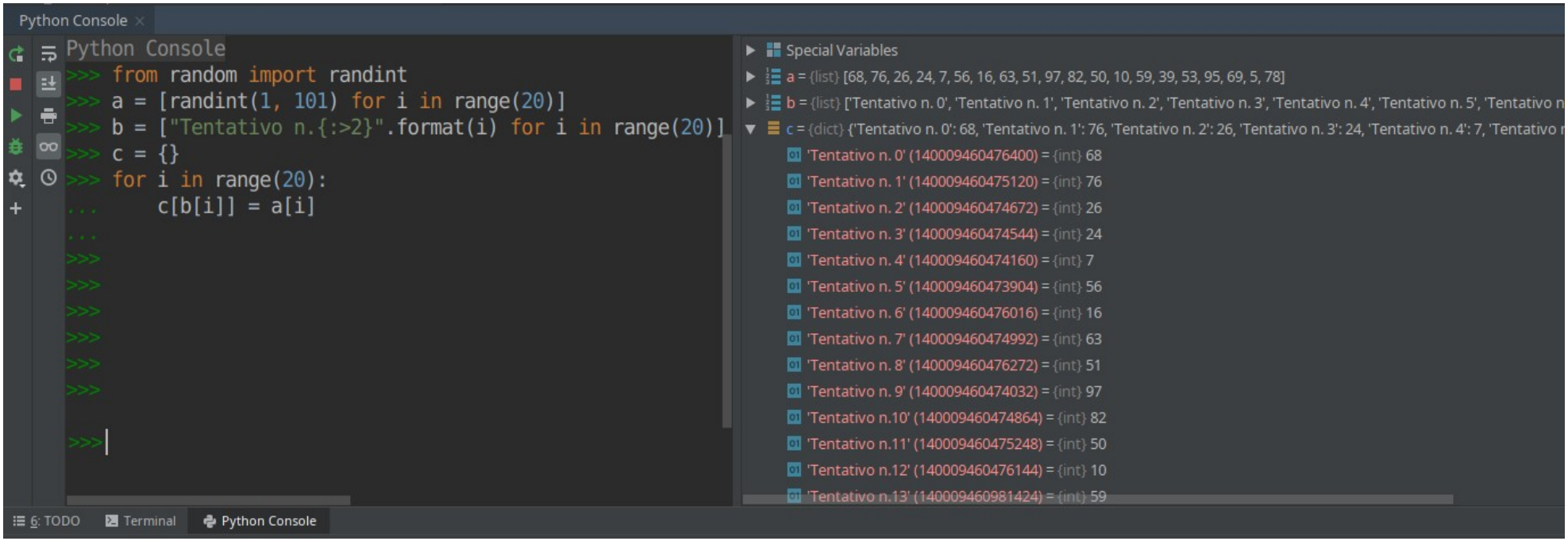
```
Python Console
>>> D = {'a': 0, 'b': 1, 'c': 3}
>>> D.values()        # è un iterabile
dict_values([0, 1, 3])
>>> 0 in D.values()  # controlla se 0 è tra i valori di D
True
>>> list(D)          # posso trasformare le chiavi del dizionario in una lista
['a', 'b', 'c']
>>> list(D.keys())    # idem
['a', 'b', 'c']
>>> list(D.values()) # posso trasformare i valori del dizionario in una lista
[0, 1, 3]
>>> list(D.items())  # posso trasformare le coppie chiave-valore in una lista
[('a', 0), ('b', 1), ('c', 3)]

>>>
```

```
Special Variables
D = {dict} {'a': 0, 'b': 1, 'c': 3}
  'a' (140686123149944) = {int} 0
  'b' (140686123444912) = {int} 1
  'c' (140686123379096) = {int} 3
  __len__ = {int} 3
```

6: TODO | Terminal | Python Console

Dizionari: creazione da due liste

Prendiamo due liste e trasformiamole in un dizionario



Dizionari: funzioni zip e dict

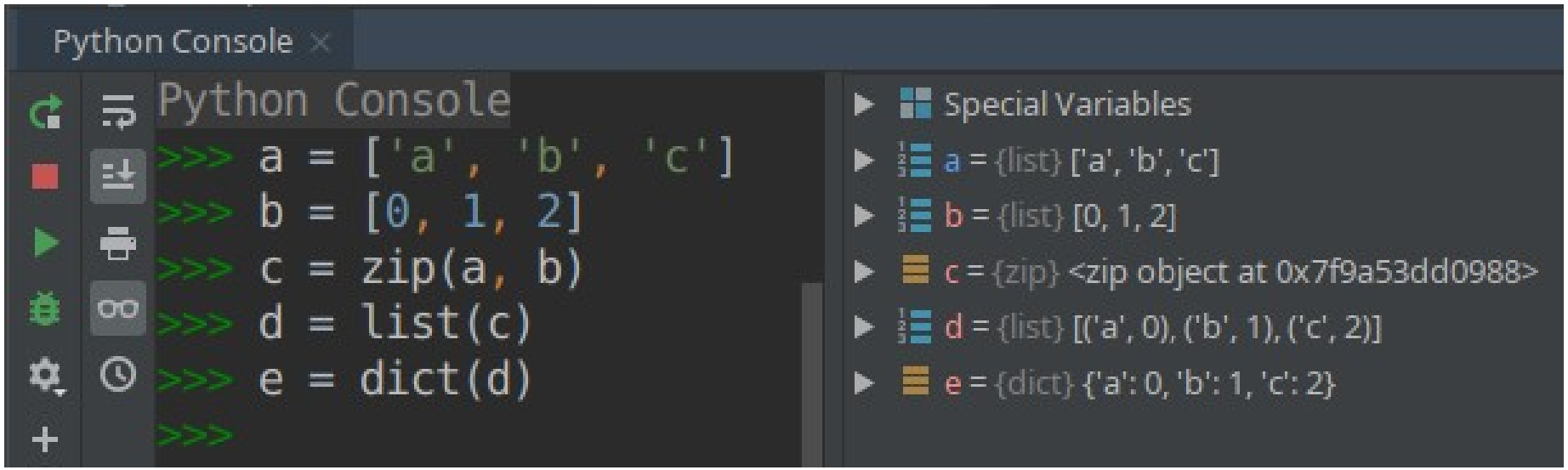
In modo più "pythonico" con le funzioni zip(seq_1, seq_2, …, seq_n) e dict()

```
Python Console
>>> from random import randint
>>> a = [randint(1, 101) for i in range(20)]
>>> b = ["Tentativo n.{:>2}".format(i) for i in range(20)]
>>> c = list(zip(b, a))
>>> d = dict(c)
>>>
```

```
Special Variables
a = {list} [25, 84, 90, 24, 85, 78, 87, 82, 50, 80, 91, 81, 57, 38, 94, 46, 36, 64, 88, 15]
b = {list} ['Tentativo n. 0', 'Tentativo n. 1', 'Tentativo n. 2', 'Tentativo n. 3', 'Tentativo n. 4', 'Tentativo n. 5', 'Tentativo n. 6',
c = {list} [('Tentativo n. 0', 25), ('Tentativo n. 1', 84), ('Tentativo n. 2', 90), ('Tentativo n. 3', 24), ('Tentativo n. 4', 85), ('Tentat
d = {dict} {'Tentativo n. 0': 25, 'Tentativo n. 1': 84, 'Tentativo n. 2': 90, 'Tentativo n. 3': 24, 'Tentativo n. 4': 85, 'Tentativo n.
randint = {method} <bound method Random.randint of <random.Random object at 0x55cb32706128>>
```

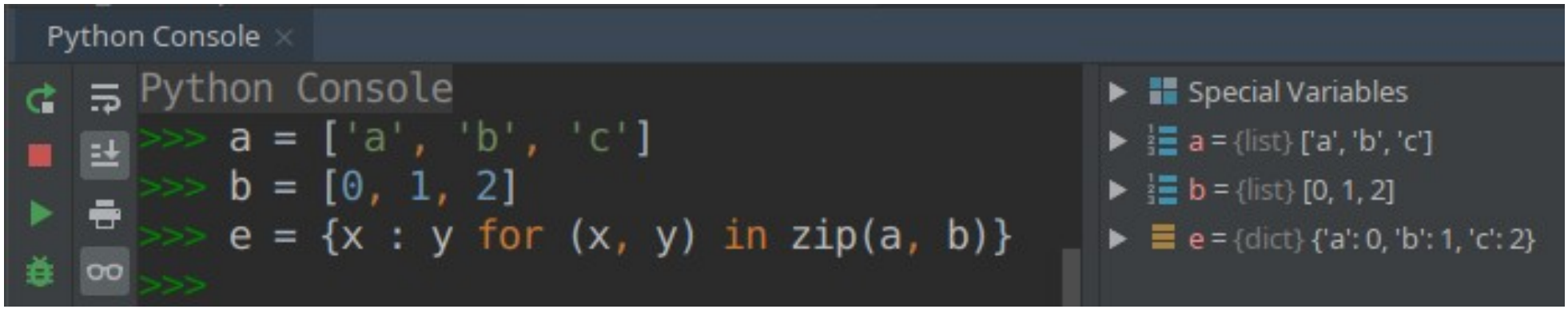
Dizionari: funzioni zip e dict

In Python3, "zip" produce, a partire da due sequenze, un iterabile, con tuple di 2 elementi; la funzione "dict" trasforma l'iterabile in un dizionario

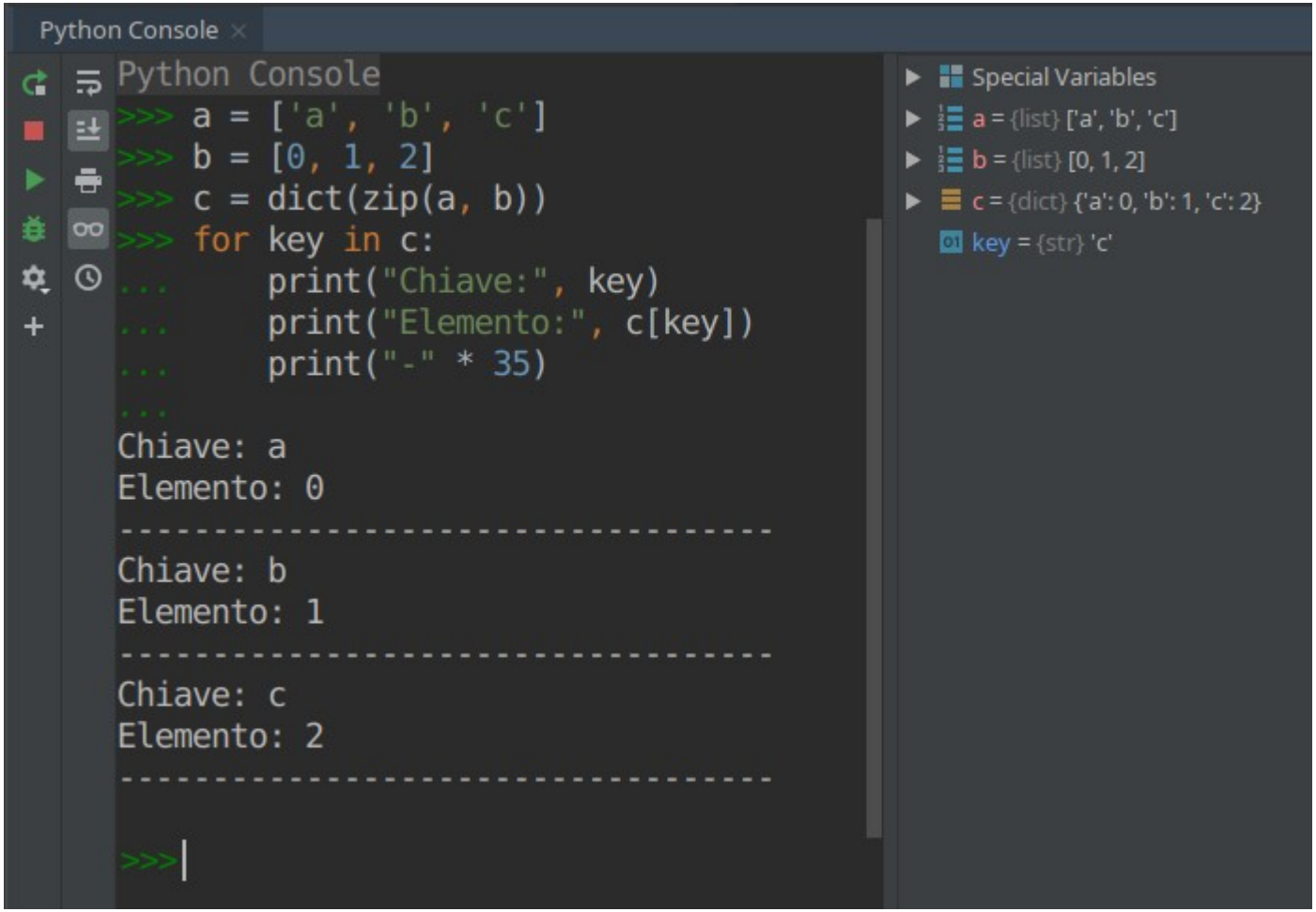


Dizionari: dictionary comprehensions

In Python esiste una "dictionary comprehensions", analoga alla "list comprehension" delle liste, che si può usare per definire gli elementi di un dizionario



Dizionari: visitare un dizionario



Un dizionario, essendo un oggetto iterabile per chiave, può essere visitato, iterando su di essa.

File: cosa sono

Un file è una sorgente (o un deposito) di informazioni, che si può leggere e scrivere; questa sorgente/deposito ha anche delle "proprietà" (nome, estensione, flag) che possono essere modificate. Questa definizione si adatta molto bene a molte periferiche ed interfacce hardware; i sistemi operativi Unix e derivati (Unix-like) hanno generalizzato il concetto di file tanto da farne una vera filosofia: in Unix tutto è un file, cioè può essere "aperto", "chiuso", "letto", "scritto" eccetera; questi "file speciali" possono essere delle comunicazioni fra processi, delle pipe, delle interfacce hardware o altro ancora: il sistema operativo si occuperà di gestire tutto nel modo appropriato lasciando ai programmi l'illusione di stare usando un normale file.

File: cosa sono

Se dal punto di vista dell'utente un file è solitamente un singolo elemento, può succedere invece che fisicamente sia scritto o risieda su più zone diverse del supporto di memorizzazione che lo ospita: questo fenomeno è molto comune se il supporto di memorizzazione è un disco di memoria, mentre è molto raro su nastri magnetici. Uno dei compiti del sistema operativo è rendere trasparente alle applicazioni la reale suddivisione fisica del file e occuparsi di gestire il recupero delle informazioni in esso contenute (lettura) dai vari blocchi fisici del supporto di memorizzazione e la scrittura.

File: a cosa servono

In certe situazioni può capitare di dover ricorrere all'aiuto di un file esterno per memorizzare dei dati, siano questi dati permanenti che verranno utilizzati in un'esecuzione successiva del programma, sia che si tratti di file temporanei.

Vediamo come sia estremamente semplice poter scrivere e leggere un file con Python.

File: funzioni e metodi

Per accedere ad un file si usa la funzione "open", che crea un oggetto file e ritorna un riferimento ad esso.

L'oggetto file ha funzioni per accedere al contenuto del file. Ci sono anche funzioni per leggere il file tutto in una volta e metterne il contenuto in una stringa, e funzioni per farne una lista di stringhe con le singole linee.

File: funzioni e metodi

L'output e' bufferizzato, cioè non scritto subito sul file, ma posto un un'area di memoria apposita (buffer) e scaricato sul file, tutto insieme, in un secondo momento, per ottimizzare i tempi di calcolo. La funzione flush scarica il buffer.

Il file ha un puntatore che ricorda dove si e' arrivati nella lettura, che, a diversi comandi di lettura, si sposta in avanti nel file. La funzione seek sposta il puntatore, permettendo di saltare parti del file o di rileggere contenuti già letti.

Il file viene chiuso con la funzione close, che elimina il riferimento al file.

File: open

```
open(nome_file, mode='r', buffering=-1, encoding=None,
      errors=None, newline=None, closefd=True)

mode e' il modo do accesso:
          'r' : per lettura di testo
          'w' : per scrivere testo
          'a' : aggiunge testo a fine file
          '+' : sia lettura che scrittura
          'b' : file con dati binari,
                ove NON si interpretano i caratteri
                come codifiche UTF-8

          I caratteri possono essere combinati, ad esempio
          'wb' ed 'rb' per scrivere o leggere dati binari.
```

File: open

```
Gli altri parametri sono raramete usati:

        buffering : da'indicazioni sull'uso e dimensione del buffer
        newline   : indicazioni sul carattere di fine linea
        errors    : su come gestire errori di codifica
        closefd   : gestisce chiusura del file

        encoding: il tipo di codifica per i caratteri: Python
                  prevede parecchie codifiche oltre UTF_8: ascii, latin-1,
                  cyrillic,  greek, UTF_16, UTF_32. In genere
                  assume la codifica di default del computer che si
                  sta usando.
```

File: open e funzioni

Esempi e funzioni per i files:

```
f= open('filetest','w') : apre un file di nome filetest per scriverci.
                          e crea un oggetto f, di tipo "file"

f1=open('filetest2','r') : apre un file di nome filetest per leggerlo
f2=open('filetest3','r+'): per leggere e scrivere
f3=open('filetest4','a') : per aggiungere in fondo al file

f,f1 etc. sono riferimenti ai files, che sono oggetti.
filetest, filetest2 etc. sono i nomi dei files.

stringa1=f1.readline()   :   legge una linea dal file f1 e la mette in stringa1
stringa2=f1.read()       :   mette in stringa2 tutto il file f1
```

File: open e funzioni

Esempi e funzioni per i files (1)

```
f= open('filetest','w') : apre un file di nome filetest per scriverci.
                          e crea un oggetto f, di tipo "file"

f1=open('filetest2','r') : apre un file di nome filetest per leggerlo
f2=open('filetest3','r+'): per leggere e scrivere
f3=open('filetest4','a') : per aggiungere in fondo al file

f,f1 etc. sono riferimenti ai files, che sono oggetti.
filetest, filetest2 etc. sono i nomi dei files.

stringa1=f1.readline()   :   legge una linea dal file f1 e la mette in stringa1
stringa2=f1.read()       :   mette in stringa2 tutto il file f1
```

File: open e funzioni

Esempi e funzioni per i files (2)

```
stringa2=f1.read(10)      :   mette in stringa2 10 byte del file f1

stringhe=f1.readlines()   :  legge tutto il file e ne fa una lista
                             ogni elemento della lista e' una linea del file

f.writelines(stringhe)    : scrive, di seguito, le stringhe della
                            lista. Per essere scritte su diverse linee le
                            stringhe devono finire con: "\n"

f.write('stringa')        : scrive sul file una stringa
f.write('stringa\n')      : scrive sul file una linea
                            (\n e' il carattere di fine linea)
```

File: open e funzioni

Esempi e funzioni per i files (3)

```
f.write('stringa')          : scrive sul file una stringa
f.write('stringa\n')        : scrive sul file una linea
                              (\n e' il carattere di fine linea)

f.flush()                   : svuota il buffer, scrivendolo tutto sul file

f.seek(5)                   : si posiziona al sesto byte del file
                              ( i bytes si contano a partire da 0)

f.tell()                    : dice a che byte e' posizionato il file

f.truncate(m)               : tronca il file dopo m bytes
```

File: open e funzioni

Esempi e funzioni per i files (4)

```
f.close()                    : chiude il file. L'oggetto 'f' viene distrutto

f.name                       : contiene il nome del file

f.mode                       : stringa che specifica il modo di accesso: 'r','w' etc.

data = open('data.bin', 'rb').read() : lettura file binario,
                                       messo in 'data' come insieme di bytes
                                       ad esempio: b'bbbbcccc\ndddd\n'
```

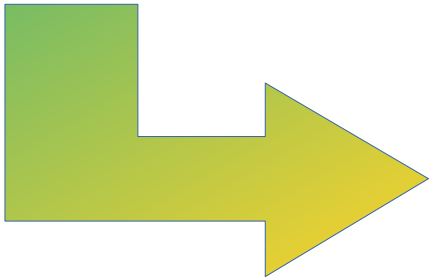
File: open e funzioni

Ci sono sistemi per iterare sulle linee del file, in modo da elaborarne una riga per volta; inoltre, i files possono essere usati in list comprehension per creare liste:

```
for line in open('data.txt'): print(line)

lines = [line.rstrip() for line in open('script2.py')]

lines= [ line.split() for line in open('script2.py')]

In quest'ultimo esempio si fa una lista di liste ove si
separano le parole di ogni linea, che vengono a costituire una
lista.
```

File: esempio di scrittura

```
1  from random import randrange
2
3  a = ["0x{:02x}".format(randrange(256)) for i in range(10)]
4  f = open("prova.txt", "w")
5  for x in a:
6      s = x + "\r\n"
7      f.write(s)
8  f.close()
9
```



prova.txt

```
1  0xff
2  0x56
3  0xce
4  0xbe
5  0x66
6  0xe4
7  0x54
8  0xa5
9  0x75
10 0xa1
11
```

File: esempio di lettura

```
0xe9
0x1d
0x7b
0xa6
0xdf
0x4e
0x36
0x10
0x36
0x6c
```

prova.txt

```
f = open("prova.txt")
for x in f:
    print(x.strip())
f.close()
```

```
0xe9
0x1d
0x7b
0xa6
0xdf
0x4e
0x36
0x10
0x36
0x6c
```

stdout

Alcuni esercizi

1) Utilizzare la list comprehension per creare la lista dei primi dieci cubi: a = [0, 1, 8, 27…]; si creino, successivamente, due nuove tuple derivate dalla precedente che contengano:
   - solo numeri pari
   - gli elementi della lista di partenza moltiplicati per 3
2) Realizzazione di un programma che data una frase in ingresso, restituisca una nuova frase le cui singole parole siano lo schema bifronte di quelle dell'input;
   Es: input: "Mi piace tanto ITS", output: "iM ecaip otnat STI"
3) Test di primalità su di un numero dato in input.
4) Creare una stringa di 40.000 caratteri scelti a caso dall'alfabeto 'CGAT'.
5) Scrivere un programma che generi 500 numeri casuali tra 0 e 255, li converta in esadecimale, con formato a due cifre, e li scriva come singole righe in un file di testo denominato "numeri.txt".
6) Scrivere un programma che legga il file dell'esercizio precedente e generi un dizionario le cui chiavi siano costituite dai numeri esadecimali letti e i valori associati alla chiave siano dati dalle occorrenze dei numeri stessi.
   Es: se nel file il numero F5 appare 13 volte il dizionario sarà così composto D = {…, "F5": 13, ...}

Il linguaggio Python (ver. 3)

Bibliografia

- Marco Beri, Programmare in Python, Apogeo
- Kenneth A. Lambert, Programmazione in Python, Apogeo
- Python Software Foundation, https://docs.python.org/3/
- Python - sito ufficiale della comunità italiana, www.python.it/doc/
- W3C programming language tutorials, www.w3schools.com/python/
- "The Hitchhiker's Guide to Python!", Kenneth Reitz, Tanya Schlusser, 2018,
  - https://media.readthedocs.org/pdf/python-guide/latest/python-guide.pdf
- www.tutorialspoint.com
  - www.tutorialspoint.com/python3/index.htm

Bibliografia

Software (free license)

- Interprete Python3: www.python.org/downloads/
- IDE: PyCharm (community): www.jetbrains.com/pycharm/download/

Immagini (free license)

- Fonte: www.duckduckgo.com