Prontuario Python: principali funzioni e metodi

Informatica (14BHD) - Anno Accademico 2022/2023 - Politecnico di Torino

Legenda (tipi degli argomenti/oggetti accettati)

`s`, `s1`: stringa
`a`, `b`, `c`, `...`: intero o float
`i`, `j`, `k`, `n`: intero
`x`: qualsiasi
`l`, `l1`: lista
`d`: dizionario
`t`, `t1`: set
`u`, `u1`: tupla
`seq`: sequenza (lista, tupla, stringa)
`cont`: contenitore (lista, tupla, stringa, set, dict)

Matematica

`abs(a)` $= |a|$
`round(a)`, `round(a, n)`: arrotonda il valore di `a` all'intero più vicino o ad `n` cifre decimali
`floor(a)`/`ceil(a)`: $\lfloor a \rfloor$ / $\lceil a \rceil$
`trunc(a)`: elimina parte frazionaria

import math ↘

`math.sin(a)`, `cos(a)`, `tan(a)`, `exp(a)`, `log(a)`, `sqrt(a)`. Possono sollevare `ValueError`
`math.isclose(a, b, rel_tol, abs_tol)`: restituisce `True` se `|a - b|` è minore o uguale di `rel_tol` (tolleranza relativa) o `abs_tol` (tolleranza assoluta).

import random ↘

`random.random()`: restituisce un numero casuale float nell'intervallo `[0,1)`.
`random.randint(i, j)`: restituisce un numero intero casuale tra `i` e `j` (estremi compresi).
`random.uniform(a, b)`: restituisce un numero reale casuale tra `a` e `b` (estremi compresi).
`random.choice(seq)`: restituisce un elemento qualsiasi della sequenza `seq`.
`random.shuffle(seq)`: rimescola in ordine casuale gli elementi della sequenza `seq`.

Operazioni comuni

`print(x, x, x, ..., sep='␣', end='\n')`: `sep` è il carattere separatore tra i valori (default spazio), `end` il carattere finale (default a capo)
`input(s)`: restituisce una stringa con le info inserite da tastiera (senza `'\n'`). `s` è il messaggio iniziale.
`range(i, j, k)`: crea una sequenza di interi che parte da `i` (compreso, default `0`), arriva fino a `j` (escluso, obbligatorio), con passo `k` (default `1`).

from pprint import pprint ↘

`pprint(...)` come `print`, ma è più ordinato (pretty print) in caso di strutture dati complesse.

Per tutti i contenitori `cont`:

`len(cont)`: restituisce il numero di elementi.
`x in cont`: restituisce `True` se l'elemento `x` è presente in `cont`, `False` altrimenti.
`sum(cont)`: restituisce la somma dei valori.
`max(cont)` / `min(cont)`: restituisce l'elemento maggiore/minore (in caso di dati strutturati, vedere `itemgetter`)
`cont.clear()`: elimina tutti gli elementi.

`sorted(cont)`: restituisce una nuova lista contenente gli elementi di `cont` ordinati. Supporta tutte le opzioni avanzate di list`.sort()`.

Per tutte le sequenze `seq`:

`seq.count(x)`: restituisce quante volte `x` è presente in `seq`.
`seq[i]`: restituisce l'elemento di indice `i` (`i<len(seq)`, altrimenti `IndexError`). Se `i<0`, parte dal fondo.
`seq[i:j]`: restituisce una sottosequenza con gli elementi consecutivi di `seq`, dalla posizione `i` (compresa, default=`0`) fino alla posizione `j` (esclusa, default=`len(seq)`).
`seq[i:j:k]`: usa `k` come "passo" per selezionare gli elementi. Se `k<0` e `i>j` va all'indietro.

Stringhe

`int(s)`: converte `s` in intero. Eccezione: `ValueError`.
`float(s)`: converte `s` in float. Eccezione: `ValueError`.
`str(x)`: converte `x` in stringa.
`ord(s)`: restituisce codice Unicode (intero) di `s[0]`.
`chr(i)`: restituisce carattere corrispondente a codice Unicode `i`. Eccezione: `ValueError`.
`s+s1`: crea e restituisce una nuova stringa concatenando due stringhe.
`s*n`: crea e restituisce una nuova stringa concatenando `n` volte la stessa stringa.
`s.lower()` / `s.upper()`: restituisce la versione minuscola/maiuscola di `s`.
`s.replace(s1, s2)` / `s.replace(s1, s2, n)`: restituisce una nuova versione di `s` in cui ogni occorrenza di `s1` è sostituita da `s2`. Se è presente `n`, sostituisce al massimo `n` occorrenze.
`s.lstrip()` / `s.lstrip(s1)`: restituisce una nuova versione di `s` in cui i caratteri di spaziatura (spazi, tab, newline) sono eliminati dall'inizio di `s`. Se è presente `s1`, vengono eliminati i caratteri presenti in essa invece dei caratteri di spaziatura.
`s.rstrip()` / `s.rstrip(s1)`: Come `lstrip`, ma i caratteri vengono eliminati dalla fine di `s`.
`s.strip()` / `s.strip(s1)`: Come `lstrip`, ma i caratteri vengono eliminati tanto a all'inizio quanto alla fine.
`s1 in s`: restituisce `True` se `s` contiene `s1` come sottostringa, altrimenti `False`.
`s.count(s1)`: restituisce il numero di occorrenze non sovrapposte di `s1` in `s`.
`s.startswith(s1)` / `s.endswith(s1)`: restituisce `True` se `s` inizia/termina con `s1`, altrimenti `False`.
`s.find(s1)` / `s.find(s1, i, j)`: restituisce il primo indice di `s` in cui inizia un'occorrenza di `s1`, oppure `-1` se non c'è. Se presenti `i` e `j`, ricerca in `s[i:j]`.
`s.index(s1)` / `s.index(s1, i, j)`: come `find`, ma se non presente solleva `ValueError`.
`s.isalnum()`: restituisce `True` se `s` contiene sole lettere o cifre e ha almeno un carattere, altrimenti `False`.
`s.isalpha()`: restituisce `True` se `s` contiene sole lettere e ha almeno un carattere, altrimenti `False`.
`s.isdigit()`: restituisce `True` se `s` contiene sole cifre e ha almeno un carattere, altrimenti `False`.
`s.islower()` / `s.isupper()`: restituisce `True` se `s` contiene sole lettere minuscole/maiuscole e ha almeno un carattere, altrimenti False.
`s.isspace()`: restituisce `True` se `s` contiene soli caratteri di spaziatura (spazi, tab e newline) e ha almeno un carattere, altrimenti `False`.

Da stringhe a liste e viceversa:

`s.split(sep, maxsplit=n)`: restituisce una lista di sotto-stringhe ottenute suddividendo `s` ad ogni occorrenza della stringa `sep` (separatore). Se `sep` è omesso, per default è una sequenza di caratteri di spaziatura. Se `maxsplit` è specificato, saranno fatte al massimo `n` separazioni partendo da sinistra (la lista avrà al più `n+1` elementi).
`s.rsplit(sep, maxsplit=n)`: come `split`, ma suddivide `s` partendo da destra.
`s.splitlines()`: come `split`, ma usa come separatore il `'\n'`, suddivide quindi `s` in una lista contenente le singole righe di testo presenti in `s`.
`s.join(l)`: restituisce una unica stringa contenente tutti gli elementi di `l` (che deve essere una lista

di stringhe) separati dal separatore `s`.

Stringhe formattate `f'{x:fmt}'`

`x` è qualsiasi variabile o espressione. `fmt` sono *codici di formattazione*, che possono contenere:
`< ^ >`: allineamento a sinistra, centrato, a destra
width: numero che indica quanti caratteri in totale deve occupare il valore. Default: quanto basta.
.precision: numero di cifre decimali (se float) o massimo numero di caratteri (se non numerico).
formato: `s` stringa, `d` intero decimale, `f` numero reale, `g` numero reale in notazione scientifica
Esempio: `f'{n:5d}␣{a:7.2f}␣{s:>10s}'`

Liste

`[]`: crea e restituisce una nuova lista vuota
`[x, ..., x]`: restituisce una nuova lista con gli elementi forniti.
`list(cont)`: restituisce una *nuova* lista contenente tutti gli elementi del contenitore `cont`.
`l * n`: restituisce una nuova lista replicando gli elementi di `l` per `n` volte.
`l + l1`: restituisce una nuova lista concatenando gli elementi di `l` ed `l1`.
`l == l1`: restituisce `True` se le due liste contengono gli stessi elementi, nello stesso ordine, altrimenti `False`.
`l.pop()`: rimuove l'ultimo elemento e lo restituisce.
`l.pop(i)`: rimuove l'elemento nella posizione `i` e lo restituisce. Gli elementi seguenti sono spostati indietro di un posto.
`l.insert(i, x)`: inserisce `x` nella posizione `i` in `l`. Gli elementi da quella posizione in poi sono spostati avanti di un posto.
`l.append(x)`: aggiunge `x` in coda alla lista `l`.
`l.count(x)`: restituisce il numero di occorrenze di `x` in `l`
`l.index(x)`: restituisce la posizione della prima occorrenza di `x` in `l`. L'elemento deve essere presente in lista, altrimenti solleva `ValueError`.
`l.index(x, i, j)`: restituisce la posizione della prima occorrenza di `x` nella porzione di lista `l[i:j]`. La posizione restituita è riferita dall'inizio della lista. Se non trovata, solleva `ValueError`.
`l.remove(x)`: rimuove l'elemento di valore `x` dalla lista e sposta indietro di un posto tutti gli elementi che lo seguono. L'elemento deve essere presente in lista, altrimenti solleva `ValueError`.
`l.extend(l1)`: aggiunge tutti gli elementi della lista `l1` alla lista `l`.
`l.reverse()`: rovescia l'ordine degli elementi nella lista `l`.
`l.copy()` o `list(l)`: restituisce una nuova lista, copia della lista `l`.
`l.sort(reverse=False)`: ordina gli elementi della lista dal più piccolo al più grande. Se si specifica `reverse=True`, ordina in ordine inverso.
`enumerate(l)`: restituisce una lista di tuple di tipo `[(indice, valore1), (indice2, valore2), ...]`, permettendo di iterare contemporaneamente su indici e valori di `l`.

from operator import itemgetter ↘

`l.sort(key=itemgetter('k'))`: ordina una lista di *dizionari* in base al valore del campo con chiave `k`. Si possono specificare anche più chiavi di ordinamento: `l.sort(key=itemgetter('k1', 'k2'))`.
`l.sort(key=itemgetter(n))`: ordina una lista di *liste* o di *tuple* in base al valore dell'elemento di indice `n`. Si possono specificare anche più chiavi di ordinamento: `l.sort(key=itemgetter(n1, n2))`. Utile anche quando la lista `l` è il risultato della funzione `enumerate()` o `dict.items()`.
`max/min(l, key=itemgetter('k'))` : in una lista di *dizionari*, restituisce l'elemento il cui valore del campo con chiave `k` è maggiore/minore.
`max/min(l, key=itemgetter(n))`: in una lista di *liste* o *tuple*, restituisce l'elemento il cui valore del campo di indice `n` è maggiore/minore. Utile anche quando la lista `l` è il risultato della funzione `enumerate()` o `dict.items()`.
Nota 1: `reverse` e `key` si possono combinare. In caso di chiavi multiple, esse vengono considerate tutte nello stesso verso (crescente o decrescente).
Nota 2: `l.sort()` e `sorted(cont)` implementano un algoritmo *stabile* (elementi con chiave

uguale mantengono l'ordine). Si possono creare ordinamenti complessi con chiamate successive di `sort`/`sorted`, dalla chiave meno importante a quella principale.

Tuple

`()`: crea e restituisce una nuova tupla vuota
`(x, ..., x)`: restituisce una nuova tupla con gli elementi forniti.
`(x,)`: nel caso di un solo elemento, è obbligatoria la virgola
Sono supportate tutte le funzioni e metodi delle liste che non modificano il valore della tupla: `u[i]`, `u+u1`, `x in u`, `u.index(x)`, `sorted(u)`, `enumerate(u)`

Insiemi

`set()`: restituisce un nuovo insieme vuoto.
`set(cont)`: restituisce un nuovo insieme che contiene una copia di `cont` (senza duplicati).
`{x, x, ..., x}`: restituisce un nuovo insieme che contiene gli elementi indicati (senza duplicati).
`t.add(x)`: aggiunge un nuovo elemento all'insieme `t`. Se l'elemento è già presente, non succede nulla.
`t.discard(x)`: elimina l'elemento dall'insieme `t`. Se l'elemento non appartiene all'insieme, non ha effetto.
`t.remove(x)`: come `discard`, ma se l'elemento non è presente solleva `KeyError`.
`t == t1`: determina se l'insieme `t` è uguale all'insieme `t1`.
`t.issubset(t1)` o `t<=t1`: determina se $t \subseteq t1$.
`t.issuperset(t1)` o `t>=t1`: determina se $t \supseteq t1$.
`t.isdisjoint(t1)`: restituisce `True` se l'intersezione degli insiemi `t` e `t1` è nulla.
`t.union(t1)` o `t|t1`: restituisce un nuovo insieme pari a $t \cup t1$.
`t.intersection(t1)` o `t&t1`: restituisce un nuovo insieme pari a $t \cap t1$.
`t.difference(t1)` o `t-t1`: restituisce un nuovo insieme che contiene gli elementi che appartengono a `t` ma non a `t1`.
`t.symmetric_difference(t1)` o `t^t1`: restituisce un nuovo insieme che contiene gli elementi presenti in uno solo degli insiemi e non in entrambi (x-or).
`t.copy()` o `set(t)`: restituisce una copia dell'insieme `t`.

Dizionari

`k` = chiave: stringa, numero, tupla
`dict()`: restituisce un nuovo dizionario vuoto.
`{}`: restituisce un nuovo dizionario vuoto.
`{k:x, ..., k:x}`: restituisce un nuovo dizionario contenente le coppie chiave/valore specificate.
`k in d`: restituisce `True` se la chiave `k` appartiene al dizionario `d`, altrimenti `False`.
`d[k] = x`: aggiunge una nuova coppia chiave/valore al dizionario `d`, se `k` non è già presente, altrimenti modifica il valore associato alla chiave `k`.
`d[k]`: restituisce il valore associato alla chiave `k`, se è presente in `d`, altrimenti solleva `KeyError`.
`d.get(k, x)`: restituisce il valore associato alla chiave `k`, se è presente in `d`, altrimenti restituisce il valore di default `x`.
`d.pop(k)`: elimina da `d` la chiave `k` e il valore ad essa associato; se non è presente, solleva `KeyError`. Restituisce il valore eliminato.
`d.items()`: restituisce una lista[a] di tuple `(k,x)` di tutti gli elementi di `d`, in ordine di inserimento.
`d.values()`: restituisce una lista[a] contenente tutti i valori presenti in `d`.
`d.keys()`: restituisce una lista[a] con le chiavi del dizionario, in ordine di inserimento.
`sorted(d)`: restituisce una lista ordinata delle chiavi del dizionario.
`sorted(d.items())`: restituisce una lista, ordinata per chiave, di tuple `(k,x)` degli elementi di `d`.
`d.copy()` o `dict(d)`: restituisce una copia del dizionario.

[a] per la precisione, restituisce una *vista*, che può essere convertita in lista con `list(...)` o che può essere iterata con un ciclo `for...in`

File

`f = open(s, modalita, encoding='utf-8')`: apre il file di nome `s`. `modalita`: `'r'` lettura, `'w'` scrittura di un file nuovo, `'a'` scrittura in coda a un file. Restituisce un "oggetto file" `f`. Eccezioni: `FileNotFoundError` se il file non esiste, in generale `OSError`.
`f.close()`: chiude il file `f`.
`f.readline()`: restituisce una stringa con i caratteri letti dal file `f` fino a `'\n'` (compreso). Restituisce `""` se a fine file.
`f.read(num)`: restituisce una stringa con (al massimo) `num` caratteri letti dal file `f`. Senza argomenti restituisce l'intero file come un'unica stringa.
`f.readlines()`: restituisce il contenuto dell'intero file sotto forma di lista di stringhe, una per riga.
`f.write(s)`: scrive `s` nel file `f`. *Nota*: non aggiunge automaticamente il fine linea `'\n'`.
`print(..., file=f)`: come `print`, ma scrive nel file `f` anziché su schermo.

import csv ↘

`csv.reader(f)`: restituisce un oggetto 'CSV reader', su cui iterare con un ciclo `for`, che restituisce ad ogni iterazione una lista i cui elementi sono i campi della prossima riga del file `f`.
`csv.DictReader(f, fieldnames=[...])`: restituisce un oggetto 'CSV dictionary reader', su cui iterare con un ciclo `for`, che restituisce ad ogni iterazione un dizionario i cui valori sono i campi della prossima riga del file `f`, e le cui chiavi sono gli elementi di `fieldnames` (o della prima riga del file, se omesso).
`csv.writer(f)`: restituisce un oggetto 'CSV writer' per il file `f`, aperto in scrittura. Si possono scrivere i dati una riga per volta usando il metodo `writerow(un_record)` oppure tutti insieme con `writerows(tutti_i_record)`.
`csv.DictWriter(f, fieldnames=[...])`: restituisce un oggetto 'CSV dictionary writer', aperto in scrittura. `fieldnames` (obbligatorio) rappresenta l'ordine e i nomi delle colonne, che devono corrispondere alle chiavi del dizionario (si può usare `d.keys()`). La prima riga del file va creata con il metodo `writeheader()`, e le righe successive, contenenti i dati, si creano con il metodo `writerow(d)`
Nota 1: i file CSV dovrebbero essere sempre aperti con l'opzione `newline=''` nella funzione `open`.
Nota 2: nel caso in cui i campi non siano separati dal carattere `','`, è possibile modificare il separatore usato, con il parametro `delimiter=';'`

import copy ↘

`copy.copy(x)`: restituisce una copia semplice ('shallow', superficiale) di `x`. Costruisce un nuovo contenitore e vi inserisce i riferimenti ai valori che erano presenti nell'originale (`x`).
`copy.deepcopy(x)`: restituisce una copia profonda ('deep') di `x`. Costruisce un nuovo contenitore e vi inserisce una nuova **copia** degli oggetti che erano presenti nell'originale (`x`) (e così via con gli oggetti in essi contenuti).

Eccezioni principali

`ValueError`: valore errato passato ad una funzione (es. `math.sqrt(-1)`) o errore nella conversione da stringa a numero (es. `int("x")`)
`IndexError`: tentativo di accesso ad una sequenza al di fuori degli indici consentiti (es. `l[len(l)]`). Attenzione: gli indici negativi (`-1`) non generano eccezione, in quando indicizzano la sequenza partendo dal fondo.
`KeyError`: tentativo di accesso ad un dizionario con una chiave inesistente.
`OSError`: errori di input-output, tipicamente nelle operazioni legate ai file, tra cui `FileNotFoundError`, `PermissionError`, `FileExistsError`.

Schema sinottico delle principali operazioni sui contenitori					
Operation	str	list	tuple	set	dict
Create	`"abc"  'abc'`	`[a, b, c]`	`(a, b, c)`	`{a, b, c}`	`{a:x, b:y, c:z}`
Create empty	`""  ''`	`[]  list()`	`()  tuple()`	`set()`	`{}  dict()`
Access `i`-th item	`s[i]`	`l[i]`	`u[i]`		`d[k]` `d.get(k,default)`
Modify `i`-th item		`l[i]=x`			`d[k]=x`
Add one item (modify value)		`l.append(x)`		`t.add(x)`	`d[k]=x`
Add one item at position (modify value)		`l.insert(i,x)`			
Add one item (return new value)	`s+'x'`	`l+[x]`	`u+(x,)`		
Join two containers (modify value)		`l.extend(l1)`		`t.update(t1)`	
Join two containers (return new value)	`s+s1`	`l+l1`	`u+u1`	`t.union(t1)  t|t1`	

Does it contain a value?	`x in s`	`x in l`	`x in u`	`x in s`	`k in d`
					(search keys)
					`x in d.values()`
					(search values)

Where is a value? (returns index)	`s.find(x)` `s.index(x)`	`l.index(x)`	`u.index(x)`		
Delete an item, by index		`l.pop(i)  l.pop()`			`d.pop(k)`
Delete an item, by value		`l.remove(x)`		`t.remove(x)` `t.discard(x)`	
Sort (modify value)		`l.sort()`			
Sort (return new **list**)	`sorted(s)`	`sorted(l)`	`sorted(u)`	`sorted(t)`	`sorted(d)` (keys) `sorted(d.items())`

vers. 3.2 - ultimo aggiornamento 13/01/2023