Complementi di Programmazione

Python: Tipi di base

CdL Informatica - Università degli studi di Modena e Reggio Emilia
AA 2023/2024

Filippo Muzzini

Tutti i tipi sono classi

Tipi Built In (alcuni godono di una sintassi agevolata)

- **Numerics: valori numerici (int, float, complex)**
- **Sequence: sequenze di oggetti (str, list, tuple)**
- **Set: insiemi (set, frozenset)**
- **Dict: dizionari di coppie key->value**

Tipi numerics

- **Numeri interi**
  - **Lunghezza arbitraria (no limiti)**
- **Numeri float**
  - **Precisione dipende dall'architettura**
  - **Separatore parte intera/decimale: “.”**
- **Numeri complessi**
  - **[Numero reale +] numero reale con suffisso j**
  - **z = 10 + 20j; z = -4j**
  - **z.real: parte reale**
  - **z.imag: parte immaginaria**
- **Booleani**
  - **Considerati sottotipo degli interi**

Tipi numerics - Operazioni possibili

- **Operazioni aritmetiche standard: +, -,*, /**
- **Divisione: / - Divisione intera: //**
  - **7.0 / 2.0 → 3.5 ; 7.0 // 2.0 → 3.0 ; 7 // 2 → 3; 7 / 2 -> 3.5**
- **Resto divisione tra interi: %**
- **Valore assoluto: abs()**
- **Numero complesso coniugato: conjugate()**
- **Elevamento a potenza: pow(), ****
- **Arrotondamento: math.trunc(), math.floor(), math.ceil(), round()**

Tipi sequenze e stringhe

Concetto di sequenza ordinata di elementi:

- **Stringhe**
  - **racchiuse tra apici ‘’ o “”**
  - **possibilità di usare i caratteri speciali es. \n \t **
- **Liste**
  - **elementi di qualsiasi tipo (anche non coerenti) racchiusi tra []**
- **Tuple**
  - **elementi di qualsiasi tipo (anche non coerenti) racchiusi tra ()**
  - **immutabili (a differenza delle liste)**

Tipi sequenze e stringhe - Operazioni

- **Accesso**
  - **accesso ad un elemento -> a[1]**
  - **accesso a sottoliste/sottostringhe -> a[1:3]**
  - **accesso partendo dal fondo -> a[-1]**
- **Concatenazione**
  - **‘a’ + ‘b’ = ‘ab’**
  - **[1,2]+[3,4] = [1,2,3,4]**
- **Ripetizione**
  - **‘a’ * 4 = ‘aaaa’**
  - **[1] * 4 = [1,1,1,1]**
- **Lunghezza della sequenza/lista**
  - **len(a) = 4**

Tipi **stringhe** - Operazioni

- **s.lower(), s.upper(): ritornano una copia della stringa s con lettere minuscole, maiuscole**
- **s.count(substr): ritorna il numero di occorrenze della sottostringa substr in s**
- **s.find(substr): ritorna l'indice della primaoccorrenza della sottostringa substr in s**
- **s.replace(sub1,sub2): rimpiazza le occorrenze della sottostringa sub1 con sub2 in s**

replace() ritorna la stringa modificata. La stringa di partenza rimane inalterata

In Python le stringhe sono immutabili

Tipi **stringhe** - Operazioni

Join:

concatena diversi elementi aggiungendo un separatore.

‘;’.join([1,2,3]) = “1;2;3”

Split:

divide una stringa in elementi considerando un separatore.

'1+2+3+4+5'.split('+’) = ['1', '2', '3', '4', '5']

Tipi **liste** - Operazioni

- lista.append(oggetto): appende l'oggetto in fondo alla lista
- lista.insert(indice, oggetto): inserisce l'oggetto nella posizione indicata dall'indice
- lista.pop(indice): estrae l'oggetto in posizione indice dalla lista
- lista.pop(): estrae l'ultimo elemento della lista
- lista.sort(): ordina gli oggetti contenuti - modifica lista in-place!
- sorted(lista) non modifica la lista originale
- len(lista): ritorna il numero di elementi contenuti in una lista
- Operatore **in**: ricerca elemento in una lista
  - 6 **in** lista → True

Tipi **liste** - Operazioni

- **lista.append(oggetto): appende l'oggetto in fondo alla lista**
- lista.insert(indice, oggetto): inserisce l'oggetto nella posizione indicata dall'indice
- **lista.pop(indice): estrae l'oggetto in posizione indice dalla lista**
- lista.pop(): estrae l'ultimo elemento della lista

Coda!

Tipi **liste** - Operazioni

- **lista.append(oggetto): appende l'oggetto in fondo alla lista**
- lista.insert(indice, oggetto): inserisce l'oggetto nella posizione indicata dall'indice
- lista.pop(indice): estrae l'oggetto in posizione indice dalla lista
- **lista.pop(): estrae l'ultimo elemento della lista**

Stack!

Tipi **liste** - Operazioni Rimozione

- lista.pop(ind1): rimuove l'elemento di indice ind1 e lo ritorna
- lista.remove(elem1): rimuove l'elemento elem1 (matching) senza ritornarlo
- del lista[ind1]: statement che rimuove l'elemento di indice ind1 – opera anche sui range!

Tipi **liste** - Operazioni Slicing

wt = [1, 2, 3, 4, 5]

- Base:
  - wt_slice = wt[1:3] -> [2,3]
  - notazione [start:stop]
- Con incremento
  - wt_slice = wt[1:5:2] -> [2,4]

Funziona anche con le stringhe!

Tipi **liste** - Copie

Se si usa **=** si copia il **riferimento**

a = [1,2,3]; b = a; b[0] =2; -> b è [2,2,3] a è [2,2,3]

E’ STATA MODIFICATA ANCHE A!

Tipi **liste** - Copie

Per copiare bisogna usare lo slicing

a = [1,2,3]; b = a[:]; b[0] =2; -> b è [2,2,3] a è [1,2,3]

B ORA E' UNA LISTA DIVERSA E OPERARE SU DI ESSA NON MODIFICA A!

Tipi **Tuple**

Le tuple sono immutabili (come le stringhe).

Si possono usare tutti gli operatori delle liste tranne quelli di modifica!

tup1 = ('one', 'two', 12, 25)

Tipi **Set**

Il set, in Python, è un insieme non ordinato di oggetti non replicati:

- Non ordinato -> Non posso accedere tramite indice
- Non replicati -> Lo stesso oggetto sarà presente al massimo una volta

Creato con la funzione **set() o {elem, elem}**.

a = set() # insieme vuoto

b = set([lista]) #insieme creato dalla lista

c = {1,2} #insieme con al suo interno gli interi 1 e 2

Tipi **Set**

I set sono comodi per certe operazioni:

- Eliminare i duplicati (per esempio partendo da una lista)
- Test di appartenenza
  - Più efficiente che scorrere una lista
  - usando l'operatore **in** come per le liste

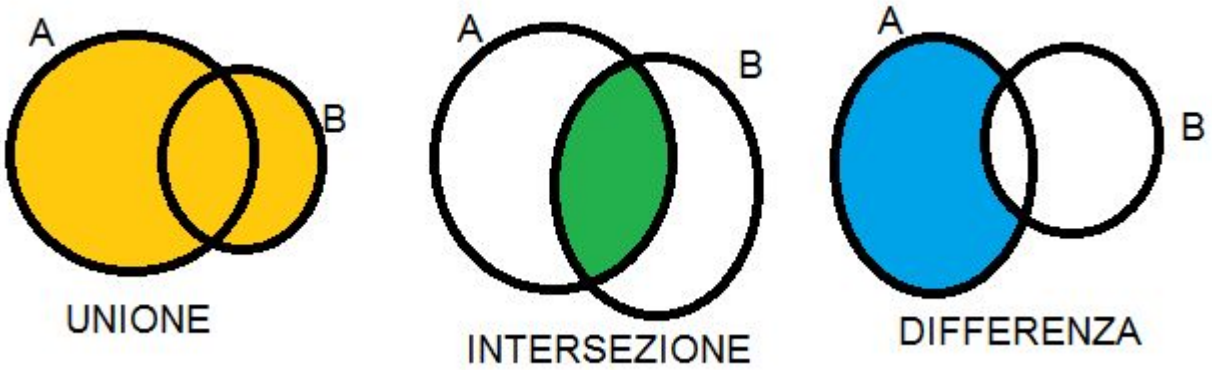
Non fatevi ingannare dalla stessa sintassi (e stessa semantica)!

L'implementazione della ricerca su set è più efficiente che su una lista

Tipi **Set** - Operazioni

- Cardinalità del set S: len(S)
- Appartenenza all'insieme: x in S, x not in S
- Disgiunzione: S1.isdisjoint(S2)
- Unione: S1.union(S2) (simbolo “|”)
- Intersezione: S1.intersection(S2) (simbolo “&”)
- Differenza: S1.difference(S2) (simbolo “-”)

Tipi **Set** - Operazioni



Tipi **Set** -. Operazioni

- Cardinalità del set S: len(S)
- Appartenenza all'insieme: x in S, x not in S
- Disgiunzione: S1.isdisjoint(S2)
- Unione: S1.union(S2) (simbolo “|”)
- Intersezione: S1.intersection(S2) (simbolo “&”)
- Differenza: S1.difference(S2) (simbolo “-”)

Tipi **Dictionary**

Il Dictionary, in Python, è un associazione **chiave-valore** di più elementi:

- simili alle HashMap in Java

Creato con **{} o dict()**.

a = {} # dictionary vuoto

a = dict() # dictionary vuoto

b = {‘chiave’: valore, ‘chiave2’: valore2} #insieme con due coppie chiave-valore

Tipi **Dictionary**

Il Dictionary, in Python, è un associazione **chiave-valore** di più elementi:

- Valori: possono essere qualsiasi oggetto
  - interi
  - stringhe
  - ecc…
- Chiavi: SOLO OGGETTI IMMUTABILI
  - Numerics
  - Stringhe
  - Tuple
- LE CHIAVI SONO UNICHE NEL DIZIONARIO

Tipi **Dictionary** - Operazioni

Accesso ai valori:

- Si accede usando la chiave con la notazione **[chiave]**

a = {‘uno’: 1, ‘due’: 2}

a[‘uno’] -> 1

Si possono anche modificare/aggiungere gli elementi in questo modo

a[‘uno’] = ‘one’ -> {‘uno’: ‘one’, ‘due’: 2}

a[‘tre’] = 3 -> {‘uno’: ‘one’, ‘due’: 2, ‘tre’: 3}

Tipi **Dictionary** - Operazioni

Metodi:

- .keys(): restituisce una lista con tutte le chiavi contenute nel dictionary
- .values(): restituisce una lista con tutti i valori contenuti nel dictionary
- .items(): restituisce una lista di tuple (chiave, valore)

Tipi **Dictionary** - Operazioni

Eliminare una chiave (e il valore):

- **del a[chiave]** : elimina la chiave e il valore dal dizionario
- .pop(chiave): elimina e restituisce il valore

Tipi **Dictionary** - Operazioni

controllo presenza chiave:

- operatore **in:** ‘uno’ **in** a -> True se la **chiave** ‘uno’ è presente in a
- .has_key(chiave): True se la **chiave** è presente nel dizionario