Riassunto comandi latex

Matteo Mistri

5 aprile 2018

Indice

Capitolo 1

Introduzione

Pyton è guidato dall'indentazione, non dalle parentesi o da ;
I commenti iniziano con #
Se scrivo invece `#! /usr/bin/env pyton3` indica che questo è un file eseguibile per il sistema operativo, ma in realtà il compilatore lo vede come un semplice commento. I tipi sono impliciti e sono bool, int, float, str non esiste char.
La divisione converte il tipo, se voglio mantenere l'intero uso la doppia barra.
Gli errorin sono gestiti come eccezioni.
Le variabili non vengono dichiarate, iniziano a esistere quando assegnate.
Gli operatori logici sono or not and, tutto in pyton è vero, tranne 0 e 0.0 e false.

Capitolo 2

Riga di comado

Per eseguire il codice(linguaggio di scriping, non serve compilare), digito:

```
phython3 nomefile.py eventualiinput
```

Capitolo 3

Liste e Iteratori

Le stringhe str sono oggetti immutabili(non si possono modificare).
Le liste sono oggetti di tipo list mutabili. Sequenze di valori anche di tipo dierso.
Le tuple sono oggetti di tipo tuple e sono immutabili. Sono come liste ma immutabili.
Gli indici partono da 0 a n-1. Posso usare indici negativi per prendere elementi a partire da -1(ultimo) a -n.
La tupla è delimitata da (), la lista da []. La tupla di un solo elemento è (elemento,).
Nelle stringhe carattere escape `\\ mentre \\n` è a capo.
Posso definire la lista anche con list(sequenzadainserire).
Se scrivo list(stringa) ho una lista con i vari caratteri.
La funzione len(variabile) prende una sequenza e restituisce la sua lunghezza.
Se scrivo seq1==seq2 ritorna true se le 2 sequenze hanno stessi elementi alle stesse posizioni.
Il metodo split(delimitatore) chiamato su una stringa torna lo split in base ai delimitatori della stringa in una lista.
La forma [espressione for variabile in sequenza] applica l'operazione a ogni elemento della lista.
Range è una funzione che prende argomenti(se ne metto 3 includo anche l'incremento) e itera su di essi. Non produce un'array di valori ma un iteratore.
Posso printare singole variabili o liste allo stesso modo.

3.1 Slicing di sequenze

- sequenza[i:j] estraggo la sottosequenza di indice i, j-1
- sequenza[i:] estraggo il suffisso da i alla fine
- sequenza[:j] estraggo dall'inizio a j-1
- sequenza[:] copio la sequenza
- seqeunza[i:j:s] estraggo da i a j-1 con salti di s
- sequenza [::-1] ho la sequenza invertita

3.2 Concatenazione

Concateno sequenze dello stesso tipo con il +
Concateno una sequenza con se stessa con il * numerodiripetizioni

3.3 Modifica Liste

Posso usare lo slicing per aggiungere elementi ad una lista.

```
lista = [1,2,3,4,5,6,7]
#7 e' la lunghezza della lista
lista[7:] = [8,9,10]
#produce una lista da 1 a 10
```

Ecco una modifca generica:

```
lista[indice] = nuovovalore
```

Posso usare anche lo slicing per modifiche di più elementi

```
del nomelista[indice]
```

Per cancellare posso anche usare come nuovo valore [ ] nella forma nomelista[indice]= [ ] del nomelista[indice]

3.4 Utilità

Il metodo rstrip() chiamato su una stringa elimina tutte le spaziature e tabulazioni.
Il metodo split(argomento) chiamato su una stringa restituisce una lista di stringhe contenente la stringa di partenza splittata secondo l'argomento passato. Se l'argomento è vuoto, viene splittata a ogni spazio o tabulazione.
Il metodo append(valore) invocato su una lista, aggiunge il valore alla lista.
Il metodo count(valore) chiamato su una lista conta il numero delle occorrenze di valore nella lista passata.
Il metodo join(listadistringhe) chiamato su una stringa, unirà le stringhe passate nella lista in un'unica stringa, usando come delimitatore la stringa sulla quale è stato chiamato il metodo.

Capitolo 4

Funzioni

Le funzioni sono dichiarate come nell'esempio e il corpo è indentato.

```
def sommatoria(var):
   totale = 0
   for elemento in var.len:
      totale += elemento
   return totale
```

Se una funzione prende n argomenti, posso passare una sequenza di n elementi scrivendo nomefunzione(*listaargomenti).

Capitolo 5

I/O

Per usare in input gli argomenti dati a linea di coamando insieme all'esecuzione dello script, accedo all'array predefinito sys.argv[indice]. Devo prima impotare il pacchetto sys. All'indice 0 c'è il nome dello script, dall'1 in poi gli argomenti.
Se devo fare I/O da file uso:

```
with open(file_name,'r') as input_file:
   for row in input_file:
      # do something with row
```

oppure, se voglio mettere l'input in un'array in cui ogni riga sia un elemento dell'array:

```
with open(file_name,'r') as input_file:
   file_rows = input_file.readlines()
```

Il metodo read() restituisce tutto il file in una sola riga, il metodo readlines restitusice una stringa per riga, inserendole in una lista

Capitolo 6

Operatore di formato

L'operatore di formato % restituisce la stringa ottenuta sostituendo i placeholders % presenti nella stringa *string name* con i valori elencati (nell'ordine) nella tupla *value tuple*
Ecco la formalizzazione seguita da un esempio praico commentato

```
'valori: %d e %d' % (34,78)

//il seguente codice restituisce i numeri decimali contenuti nella
    variabile totale con una sola cifra decimale
'%.1f' % totale
```

Capitolo 7

Collezioni

7.1 Dizionari

I dizionari sono oggetti mutabili di tipo dict. Sono una collezione di valori anche di tipo diverso, senza un'ordine o una posizione, ma sono indicizzati tramite una chiave(solitamente stringhe). Sono quindi una collezione di coppie chiave valore. Ecco come si definiscono:

```
nomedizionario = {chiave : valore, chiave2 : valore2}
nomedizionario2 = dict(lista tuple)
```

Per accedere uso nomedizionario[chiave]
Per modificare o aggiungere uso nomedizionario[chiave]=nuovovalore
elemento in nomedizionario ritorna true se elemento è chiave nel dizionario
La funzione len(nomedizionario) restituisce il numero di elementi nel dizionario
Il metodo values restituisce gli elementi in una lista
Il metodo keys restituisce le chiavi in una lista
Il metodo items restituisce le tuple con le coppie chiavi valore
Il metodo update prende una lista di tuple e aggiorna il diz tramite aggiunta o aggiornamento
Il metodo pop(chiave) ritorna un elemento del dizionario eliminandolo
Il metodo clear svuota un dizionario
Il metodo `has_key(chiave)` chiamato su un dizionario restituisce true se il dizionario contiene la chiave

7.2 Insiemi

L'insieme è una collezione di oggetti mutabili non ordinata di valori di tipo misto, senza duplicati.
Si dichiara così:

```
nomeinsieme=set(quellochevoglio)
#quello che voglio possono essere tuple o un dizionario
```

La funzione len(nomeinsieme) restituisce la lunghezza Posso usare in come sopra
Il metodo add permette di aggiungere un elemento

7.3 Contatori

Il contatore è un oggetto counter che va importato. E' come un dizionario in cui valori sono solo interi. Se lo uso su una stringa, conta le ripetizioni delle lettere. Idem per le tuple. Usa le lettere/tuple come chiave e il valore sono le ripetizioni. Si dichiara così:

```
from collections import Counter

contatore = Counter(tuple)
```

Il metodo `most_common` restituisce le coppie chiave valore in ordine di valore decrescente

Capitolo 8

Espressioni Regolari

Devo importare le re per poter utilizzare le espressione regolari e le sue funzioni. Ogni simbolo rappresenta se stesso, tranne alcuni caratteri speciali. Per rappresentare che siano letti come caratteri speciale, antepongo un \

8.1 Ancore

- `^` indica l'inizio riga
- `$` indica fine riga
- `\A` inizio stringa
- `\Z` fine stringa
- `\b` confine parola
- `\B` non confine parola

8.2 Classi

Uso le classi per indicare famiglie di caratteri al posto del singolo carattere

- `[abcdf]` contiene le lettere abcdf
- `[a-z]` contiene le lettere minuscole

- `[A-Z]` contiene le lettere maiuscole
- `[0-9]` contiene i numeri da 0 a 9
- `[a-zA-Z]` contiene tutte le lettere
- `[a-zA-Z0-9_]` contiene tutti i simboli di parola

Se metto come primo carattere il `^` nego il contenuto della classe.
L'operatore `&&` esegue l'intersezione tra classi.

8.3 Metasimboli

Sono abbreviazioni delle classi

- `\d` numeri da 0 a 9
- `\w` lettere e numeri
- `\s` spazi e tab
- `.` ogni simbolo tranne `\n`

Se utilizzo i metasimboli in maiuscolo, li nego.

8.4 Quantificatori

Posso usare i moltiplicatori + per indicare da 1 a n ripetizioni. Il carattere * indica da 0 a n ripetizioni. Il carattere ? invece permette che vi siano 0 o 1 ripetizioni del carattere. Se voglio un numero di ripetizioni definito, lo/li inserisco tra separati da virgole. Il simbolo | fornisce un'alternativa al carattere.

8.5 Matching e Searching

La funzione re.search(pattern, stringa) ritorna un oggetto contenente le info del primo matching del pattern nella stringa. Su di esso posso usare il metodo start e il metodo end per ottenere indici di inizio e fine occorrenza.

La funzione match funziona allo stesso modo, ma funziona SOLO SE LA STRINGA INIZIA CON IL PATTERN.

8.6 Back reference

Posso dividere con stratagemmi la stringa in sottopattern da richiamare. Tramite il metodo group(indice) chiamato sull'oggetto ritornato dalla search posso accedere ai vari campiin cui è divisa la sottostringa di matching.

```
string = 'gatto cane'
p = '(\w+)\s(\w+)'
m = re.search(p, string)
print(m.group())
#stampa gatto cane
print(m.group(1))
#stampa gatto
print(m.group(2))
#stampa cane
```

8.7 Metodi utili

La funzione re.findall(pattern, stringa) ritorna una lista con tutte le occorrenza del pattern
La funzione re.sub(pattern, newpattern, stringa) restituisce una stringa con tutte le occorrenze del pattern nella stringa sostituite con il nuovo pattern

Capitolo 9

Ibridazione con codice C

Ci sono operazioni che fatte in phython sono molto ingorde di risorse. Come creare funzioni c che siano efficienti in modo da fargli fare operazioni che in python sarebbero troppo dispendiose?
Creare un programma c e richiamarlo da python con una syscall mi comporta costi aggiuntivi di context switch e posso passare parametri limitati.
Posso usare al posto di un programma esterno fatto e finito, una libreria(nel file h, devono essere definite le funzioni che voglio rendere accessibile). Anche la compilazione avverrà in modo differente per i files c e h, in modo da rendere il file compilato shared.
Nello script python devo importare la libreria FFI(from cffi import FFI), creare un oggetto FFI(lib = ffi.dlopen('./nomelibreria')) e dichiarare le firme delle funzoni che ho intenzione di utilizzare (ffi.cdef('firmefunzionidausare'))
Ora posso invocare la mia funzione c semplicemente con la dot notation sull'oggetto lib.
Vi sono altri modi di risolvere il problema: posso accedere direttamente alle strutture dati del c, senza quindi bisogno che la funzione chiamata mi restituisca il risultato(evitando quindi ci costi di conversione della soluione).
Posso anche usare utilizzare Cython, una estensione di Python che è direttamenre compatibile con i tipi di dato C. Questo mi crea però possibili problemi di compatibilità e fa perdere l'immediatezza del linguaggio.

Capitolo 10

Formato FastQ

Usando le nuove tecniche di sequenziamento, ottengo un dato con qualità variabile. Nel file fastq, oltre alla sequenza è salvata la qualità di ogni singola base. Questò è molto importante per creare tools che utilizzano dati sequnziati in questo modo. Il più usateo è phred quality score (-10*log(p) dove 0<p<1 è la probabilità che la base b sia ERRATA). I valori variano tra 0 e infinito. Sopra il 30 è buona, sopra il 50 è ottima.
Il fastq ha 4 righe che rappresentao una entry. La prima inizia con @ e segue l'ID della read. La seconda, è la riga delle basi lette. La terza inizia con + e ripete l'ID della read. Quarta riga contiene la sequenza dei phread. I phread sono simboli ASCII, per ottenere il valore numerico si usa una funzione data che sfrutta il valore del simbolo ASCII. Ecco l'esempio di una enty:

```
@HWUSI-EAS522:8:5:662:692#0/1
TATGGAGGCCCAACTTCTTGTATTCACAGGTTCTGC
+HWUSI-EAS522:8:5:662:692#0/1
aaaa‘aa‘aa‘]__‘aa‘_U[_a‘^aaUTWZ‘X^QX
```

Concetto di **trimming**: i dati possono avere prefisso e/o suffisso di bassa qualità(il concetto di bassa qualità e relativo al contesto). Se quello che rimane è troppo corto, lo butto, altrimenti lo tengo. Se scelgo come soglia per esempio 58, cerco la più lunga sequenza contigua che abbia valori >58 il resto lo butto.

Capitolo 11

Allineamento e formato SAM

11.1 Allineamento

Concetto di query e reference: la query è la read prodotta da sequenziamento che viene cercata nella reference, che rappresenta il cromosoma. Esiste una versione binaria, detta BAM, più facile e veloce da gestire e indicizzare. Serve per salvare nel file diverse forma di allineamento tra query e reference:

- MAPPING: la query si allinea a una sottostringa della reference. Gli indel iniziali e finali non sono mappati, in quanto superflui
- CLIPPED ALLIGNMENT: una sottostringa della query si allinea a una sottostringa del reference. Viene quindi eliminata dalla testa e dalla coda della read una parte che non si allinea. Le parti eliminate e gli indel inizali e finali si omettono nel caso di HARD clipping, se invece non elimino le parti inizali e finali si chiama SOFT clipping
- SPLICED ALLIGNMENT: la query è una concatenazione di parti che si allineano a diverse sottostringhe della reference (per esempio mRNA che non mappa gli introni). Vengono inseriti gli indel tra le varie parti che compongono la query
- MULTIPLE ALLIGNMENT: la query si mappa in regione diverse del reference. Vengono rappresentati nel SAM in record diversi ma legati logicamente
- MULTI-PART ALLIGNMENT: la query è una concatenazione di parti che si allineano a diverse sottostringhe della reference. In output avrò più record in que viene lasciato il clipping delle varie parti

- PADDED ALLIGNMENT: ho query multiple, in caso di mapping che si sovrappongono, derivo le varie matrici di allineamento e creo la matrice di allineamento multiplo

11.2 Formato Sam

Ho due sezioni, una di header e una di allignment. Nella sezione di header ho record che iniziano con @ID seguito dagli attributi. Ecco alcuni esempi:

- @HD header
- @SQ reference(posso averne più di 1)
- @RG gruppo di reads
- @PG software di allineamento
- @CO commenti

Nella sezione di allignment, ogni riga è un read. Ogni read ha undici campi separati da tab obbligatori più due campi opzionali.

1. ID della read
2. numero a 16 bit che rappresenta un arrat di flag con significati specifici
3. id della reference a cui il read si allinea
4. posizione sulla reference dell'inizio dell'allineamento
5. qualità dell'allineamneto $-10 * log_10(pw)$
6. stringa di cifre intere e caratteri maiuscoli che indicano le info per derivare la matrice di allineamneto tra query e reference
7. identificatore della reference sequence dell'allineamento primario del mate read (se esiste)
8. posizione di inizio dell'allineamento del mate read
9. inferred insertion size
10. sequenza primaria del read
11. sequenza di qualità del read, come nel fastq

La codifica del sesto campo segue queste regole:

M: match/mismatch
I: inserimento nella reference
D: delezione nella reference
N: inserimento nella reference dovuto ad allineamento spliced(dovuto ad un introne)
S: soft clipping (della sequenza di query)
H: hard clipping (della sequenza di query)
P: "delezione" silente (padding)

Il numeo che precede la lettera rappresenta il numero di operazioni consecutive fatte. Per esempio:

acgtgtga—-gcgtaacgtggcaaa
—-gtgattgcg————

POS = 5
CIGAR=4M2D3M
SEQ= gtgattgcg