Laboratorio d'Informatica

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

A.A. 2021/2022

Docente: Lorenzo Putzu

Lezione 05

Il linguaggio Python

Le liste: funzioni iterative

- L'uso di variabili per rappresentare gli indici di una lista consente di esprimere in modo conciso, attraverso un'istruzione iterativa, la ripetizione di una stessa sequenza di istruzioni su tutti gli elementi di una lista.
- Questo è possibile usando una variabile nel ruolo di indice e la funzione len, che consente di applicare la sequenza di istruzioni anche a liste delle quali non è nota la lunghezza nel momento in cui si scrive il programma.

```
k = 0
while k < len(lista):
    istruzioni che coinvolgono lista[k]
    k = k + 1
```

- Notare anche la condizione k < len(lista) nell'istruzione while: il valore dell'indice dell'ultimo elemento è infatti pari alla lunghezza della lista **meno uno**.

Le liste: funzioni iterative

- Come altri linguaggi, anche Python include una versione alternativa dell'istruzione iterativa while che è l'istruzione for.
- Nel caso di Python l'istruzione for consente di esprimere un iterazione in modo molto più conciso accedendo iterativamente a **tutti** gli elementi di una sequenza (stringa o lista)
- Sintassi

```
for v in s :
    sequenza di istruzioni
```

- v deve essere il nome di una variabile che non sia già usata per memorizzare dati all'interno dello stesso programma
- s deve essere un'espressione avente come valore una sequenza (una lista o una stringa)
- sequenza di istruzioni è una sequenza di una o più istruzioni qualsiasi che di norma eseguono un'operazione sulla variabile v

Le liste: funzioni iterative

- La sequenza di istruzioni viene eseguita per un numero di volte pari alla lunghezza di s.
- Nella *i*-esima iterazione, prima dell'esecuzione di sequenza di istruzioni viene assegnato alla variabile v il valore dell'*i*-esimo elemento di s; la sequenza di istruzioni può quindi elaborare tale valore accedendo a v.
- L'istruzione for non richiede l'uso esplicito degli indici per accedere agli elementi di una sequenza, controllo che con l'istruzione while è demandato al programmatore
- Esempio

```
lista = list(range(0, 50, 5))
print("I suoi elementi sono:")
for elemento in lista :
    print(elemento)
```

Le liste: funzioni iterative

- Tuttavia, v contiene una **copia** degli elementi di s: questo significa che, se s è una lista, non sarà possibile modificare i suoi elementi assegnando nuovi valori a v.

- Non sarà comunque possibile **modificare** il valore di un elemento di una lista, attraverso un'istruzione di assegnamento che richieda l'uso esplicito degli indici come lista[k] = valore

- Non si potrà accedere nello stesso passo di un'iterazione a **più di un elemento** di una sequenza, per esempio per confrontare i valori di due elementi adiacenti come if s[k] != s[k + 1] :

Le liste: funzioni iterative

- È comunque possibile, anche con l'istruzione for, accedere agli elementi di una sequenza s usando una variabile come indice: a questo scopo l'istruzione for deve essere applicata non a s, ma ad una sequenza s2 che contenga i valori degli indici di s, nell'ordine desiderato, secondo lo schema seguente:

```
for k in s2 :
    istruzioni che accedono a s[k]
```

- In particolare, se si vuole accedere a s dal primo all'ultimo elemento, la sequenza s2 dovrà contenere gli interi da 0 alla lunghezza di s meno uno, e quindi può essere ottenuta usando la funzione predefinita range già vista in precedenza:

```
for k in range(len(s))
    istruzioni che accedono a s[k]
```

Le liste: funzioni iterative

- Un'ulteriore alternativa consiste nell'uso della funzione predefinita di Python `enumerate`, che permette di scorrere contemporaneamente gli elementi della lista e i corrispondenti indici
- Sintassi

  for i, v in enumerate(s) :

  sequenza di istruzioni

- i contiene gli indici corrispondenti agli elementi v nella sequenza s
- v come prima contiene una **copia** degli elementi di s

- Esempio

```
for i, v in enumerate(lista) :
    print("indice :", i)
    print("elemento :" ,v)
```

Esercizi

1. 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
2. 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`
3. 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]

Le liste: ulteriori funzioni predefinite

Altre funzioni (metodi) di utilità generale sono le seguenti:

- `lista.append(espressione)`
  aggiunge espressione come ultimo elemento di lista
- `lista.insert(index,espressione)`
  aggiunge espressione in lista alla posizione indicata da index
- `lista.index(espressione)`
  se espressione esiste restituisce il suo indice altrimenti da un errore
- `lista.count(espressione)`
  restituisce il numero di volte in cui espressione appare nella lista

Le liste: ulteriori funzioni predefinite

Altre funzioni (metodi) di utilità generale sono le seguenti:

- `lista.remove(espressione)`
  se espressione esiste lo rimuove dalla lista altrimenti da errore
- `lista.pop(index)`
  rimuove e restituisce l'elemento alla posizione indicata da index
- `lista.clear()`
  svuota la lista
- `lista.sort()`
  riordina la lista in ordine crescente
- `lista.reverse()`
  Inverte l'ordine degli elementi della lista

Le liste e le stringhe: similitudini

- Le stringhe e le liste sono tipi di dato che hanno in comune il fatto di essere costituite da **sequenze ordinate** di un numero qualsiasi di elementi. Per questo motivo sono entrambe indicate in linguaggio Python con il termine più generale di sequenze.
- Alcuni operatori e alcune funzioni predefinite che operano su sequenze ordinate possono essere applicati sia alle liste che alle stringhe (come si è già visto per l'operatore di concatenazione):
  - operatori: in e not in, indicizzazione, *slicing*

Le liste e le stringhe: differenze

- Liste e stringhe presentano però una differenza fondamentale:
  - attraverso l'istruzione di assegnamento e l'operatore di indicizzazione si è visto che è possibile **modificare** i singoli elementi di una lista: le liste sono perciò dette sequenze mutabili
  - non è invece possibile modificare i singoli caratteri delle stringhe, che per questo sono dette immutabili (il tentativo di modificare un elemento di una stringa produce un errore)

Le stringhe

- Esistono delle ulteriori funzioni predefinite per le stringhe che sono molto utili per la loro eleborazione, in particolare quando si acquisiscono stringhe da tastiera o (come vedremo più avanti) da *file* di testo.
- La funzione predefinita split ad esempio suddivide una stringa in corrispondenza dei caratteri di spaziatura (incluso il *newline*) e restituisce una **lista** contenente le corrispondenti sottostringhe, nelle quali non vengono inclusi i caratteri di spaziatura (la stringa originale **non viene modificata**)
- La sintassi della chiamata è diversa da quella vista in precedenza (vedremo in seguito il motivo)
- Sintassi

```
stringa.split()
```

- Ad esempio con le istruzioni

```
stringa = "Questa è una frase."
stringa.split()
```

- daremo vita ad una lista con i seguenti elementi ["Questa", "è", "una", "frase."]

Le stringhe

- É inoltre possibile suddividere una stringa in corrispondenza di una sequenza di uno o più caratteri qualsiasi che non siano il carattere di spaziatura o il newline, tale sequenza dovrà essere indicata (sotto forma di **una** stringa) come argomento di split, con la seguente sintassi:

```
stringa.split(caratteri)
```

- Per esempio, se la variabile stringa contenesse una stringa composta da una sequenza di numeri separati da una virgola (senza spazi), potremmo suddividere la stringa con le seguenti istruzioni

```
stringa = "15,1,25,9,6,21"
stringa.split(",")
```

- otterremo la lista ["15", "1", "25", "9", "6", "21"]

Le stringhe: funzioni predefinite

- Alcune funzioni predefinite di utilità generale sono le seguenti:
  - `len(stringa)`
    restituisce la lunghezza di una stringa
  - `str(espressione)`
    restituisce espressione sotto forma di stringa
  - `stringa.count(stringa2)`
    restituisce il numero di volte in cui stringa2 appare nella stringa
  - `stringa.find(stringa2)`
    restituisce la posizione in cui stringa2 appare per la prima volta nella stringa
  - `stringa.isnumeric()`
    restituisce True se tutti i caratteri nella stringa sono numerici

Le stringhe: funzioni predefinite

- Altre funzioni predefinite utili per l'elaborazione e il confronto tra stringhe:
  - `stringa.replace(stringa2,stringa3)`
    rimpiazza le occorrenze di stringa2 con stringa3
  - `stringa.lower()`
    converte la stringa in minuscolo
  - `stringa.upper()`
    converte la stringa in maiuscolo
  - `stringa.capitalize()`
    converte il primo carattere della stringa in maiuscolo

Esercizio

4. Modificare il programma della lezione precedente (Lez04_es01)
   - aggiungendo un ulteriore funzione per il calcolo della circonferenza dato il raggio n
   - richiedere un'ulteriore parametro di tipo stringa all'utente in modo che possa decidere che calcolo effettuare, quindi i possibili valori potrebbero essere: circonferenza, fattoriale e serie armonica
   - modificando la funzione per il controllo del numero inserito dall'utente, in modo che verifichi se si tratta di un numero intero non negativo n se il calcolo richiesto è uno tra fattoriale e serie armonica, oppure possa essere un float o intero se il calcolo richiesto è la circonferenza

   - Plus: l'utente potrebbe anche chiamare la funzione principale (ad esempio 'calcola') come segue

     >>> calcola('fattoriale',6)