Laboratorio di Informatica

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

A.A. 2021/2022

Docente: Lorenzo Putzu

Lezione 03

Il linguaggio Python

Istruzione break

- Un'istruzione iterativa termina quando, all'inizio di un'iterazione, l'espressione condizionale risulta falsa.
- Per concludere l'esecuzione di un'istruzione iterativa è anche possibile usare l'istruzione `break`. Questa istruzione può essere usata **solo** all'interno di un'istruzione iterativa, in un qualsiasi punto della sequenza di istruzioni da ripetere.
- Di norma l'istruzione break viene scritta all'interno di un'istruzione condizionale nidificata in un'istruzione iterativa, per concludere l'esecuzione di quest'ultima nel caso in cui si verifichi una data condizione.
- Quando l'interprete incontra l'istruzione break conclude immediatamente l'esecuzione dell'istruzione iterativa e passa a eseguire l'istruzione successiva.

Istruzione break

- L'esempio di prima può quindi essere riscritto come segue

```
x = eval(input("Inserire un numero: "))
k = 1
while True:
    print("Buongiorno")
    if k > x :
        break
    k += 1
```

Esercizi

- Creare una nuova versione degli esercizi fatti nella lezione precedente
  1. Lez02_es02
  2. Lez02_es03

  in modo che includano un'istruzione `break`

Commenti

- È sempre utile documentare i programmi inserendo **commenti** che indichino quale elaborazione viene svolta dal programma, quali sono i dati di ingresso e i risultati, qual è il significato delle variabili o di particolari blocchi di istruzioni, ecc.

- Nei programmi Python i commenti possono essere inseriti in qualsiasi riga, preceduti dal carattere # (“cancelletto”) e tutti i caratteri che seguono il cancelletto, **fino al termine della stessa riga**, sono considerati commenti e vengono trascurati dall’interprete.

```
#questo è un commento
print('stampa qualcosa') # anche questo è un commento
```

Commenti

- I commenti si possono estendere anche su più righe e per questo scopo si utilizzano i tripli apici ad entrambe le estremità

```
'''
Questo è un commento che si estende su più linee
'''
```

- Questo tipo di commenti si utilizza per una descrizione più generica e globale di una serie di istruzione che seguono

Cast implicito ed esplicito

- Spesso capita di fare delle operazioni tra tipi di dati differenti di dati, ad esempio tra interi e floats. In molti casi l'interprete python converte implicitamente i dati e si parla appunto di **cast implicito**

- Ad esempio il risultato prodotto dall'operatore / è sempre rappresentato come valore frazionario anche quando i valori sono entrambi interi
  - 5 / 2 = 2.5
  - 4 / 2 = 2.0

- Spesso è necessario e preferibile convertire direttamente le variabili prima di effettuare delle operazioni e lo si può fare con la seguente sintassi

```
int(espressione)
float(espressione)
```

Cast implicito ed esplicito

- int(numero)

  restituisce la parte intera di un numero

- int(stringa)

  se stringa contiene la rappresentazione di un numero **intero**, restituisce il numero corrispondente a tale valore

- float(numero)

  restituisce il valore di numero come numero frazionario (*floating point*)

- float(stringa)

  se stringa contiene la rappresentazione di un numero qualsiasi (sia **intero** che **frazionario**), restituisce il suo valore espresso come numero frazionario

Migliorare la leggibilità

- Nel linguaggio Python è possibile suddividere **una** istruzione in più righe, inserendo il carattere \ (*backslash*) nel punto in cui si interrompe una riga.
- Questa possibilità può essere sfruttata per suddividere in due righe un'espressione condizionale troppo lunga o anche l'esprezzione presente in un'istruzione print.
- Nella prosecuzione di una riga si può inserire un rientro qualsiasi (anche nessuno). Per garantire la leggibilità del programma è però buona norma usare un rientro coerente con il contenuto della riga precedente.

```
if  x >10:
    print('stavo scrivendo una stringa da stampare a schermo ma mi \
    sono accorto che era troppo lunga e peggiorava la leggibilità')
```

Struttura dei programmi

- In generale un programma prevede:
  - l'acquisizione dei dati da elaborare (attraverso la tastiera con la funzione input, o con altri meccanismi che saranno presentati più avanti) e il loro assegnamento a opportune variabili
  - l'elaborazione dei dati d'ingresso e degli eventuali risultati intermedi fino a ottenere i risultati desiderati, per mezzo di opportune sequenze di istruzioni (in particolare, i risultati intermedi dovranno essere memorizzati in opportune variabili)
  - la stampa dei risultati sullo schermo per mezzo della funzione print (oppure il loro invio ad altri dispositivi periferici, come si vedrà più avanti)

Esercizi

3. Calcolare il fattoriale di un dato numero non negativo n:

$$n! = 1 \times 2 \times \cdots \times (n-1) \times n, \text{ se } n > 0$$
$$n! = 1, \text{ se } n = 0$$

4. Calcolare la somma dei primi n termini della serie armonica per un dato intero non negativo n:

$$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

5. Per un dato $x \geq 0$ trovare il più piccolo intero n tale che:

$$\sum_{k=1}^{n} \frac{1}{k} > x$$

Funzioni predefinite

- Finora abbiamo visto e utilizzato delle **funzioni** messe a disposizione del programmatore come parte integrante dell'ambiente di programmazione, che non sono altro che un insieme di programmi che eseguono operazioni di utilità generale

- Sono dette funzioni predefinite, o *built-in* e l'insieme di tali funzioni viene detto libreria.

- Sono caratterizzate dalla capacità di elaborare un determinato insieme di valori di ingresso e di produrre un risultato in uscita
  - La disponibilità di tali funzioni evita agli utenti di dover scrivere propri programmi per realizzare operazioni di utilità generale
  - Ci possiamo focalizzare su **COSA** fanno piuttosto che sul **COME** lo fanno

Funzioni predefinite

- Il linguaggio Python comprende un vasto insieme di funzioni di libreria, disponibili in qualsiasi ambiente di programmazione. Alcune funzioni predefinite sono direttamente accessibili dalla *shell* e dai propri programmi. Le altre sono suddivise in diverse categorie, dette a loro volta *librerie*
  - La descrizione completa è disponibile nel sito http://docs.python.it
  - Anche se utilizzeremo principalmente
    - le funzioni matematiche
    - le funzioni per la generazione di numeri casuali

Funzioni predefinite

- A ogni funzione è associato un nome e può elaborare un determinato numero di valori di ingresso, detti argomenti e restituire un valore di un determinato tipo, come **risultato** dell'elaborazione svolta
- Ogni volta che abbiamo la necessità di eseguire una funzione effettuiamo una chiamata di funzione
- **Sintassi**: res = nome-funzione (arg1, arg2, . . . , arg*n*)
  - arg1, . . . , arg*n* sono **espressioni** Python, i cui **valori** costituiranno gli argomenti della funzione il cui tipo e numero dipende dalla specifica funzione:
  - il tipo e il numero di argoment deve coincidere con quello previsto dalla funzione altrimenti si otterrà un messaggio d'errore
  - res costituisce il valore restituito dalla funzione

Funzioni predefinite

- Come casi particolari, esistono anche funzioni che
  - non ricevono **nessun** argomento
  - non restituiscono **nessun** risultato
  - possono gestire un numero di argomenti che non è determinato a priori

Funzioni predefinite

- Come casi particolari, esistono anche funzioni che
  - non ricevono **nessun** argomento
  - non restituiscono **nessun** risultato
  - possono gestire un numero di argomenti che non è determinato a priori

- La funzione input può non ricevere argomenti, oppure può riceverne uno
  - input() oppure input(arg)
  - restituisce un valore di tipo **stringa**.

- La funzione print consente di stampare nella *shell* i valori di una o più espressioni:
  - print(espressione) oppure print(espressione1,espressione2,. . .)
  - non **restituisce** nessun valore: il suo unico scopo è **stampare** valori nella *shell*.

Funzioni predefinite

- Altre funzioni predefinite di utilità generale sono le seguenti:
  - `len(stringa)`

    restituisce il numero di caratteri di una stringa
  - `abs(numero)`

    restituisce il valore assoluto di un numero
  - `str(espressione)`

    restituisce una stringa composta dalla sequenza di caratteri corrispondenti alla rappresentazione del valore di espressione (che può essere di un qualsiasi tipo: numero, stringa, valore logico, ecc.)

Funzioni predefinite

- La chiamata di una funzione è un'**espressione**, e quindi deve rispettare le stesse regole sintattiche:
  - può comparire come operando di una qualsiasi espressione più complessa
  - può comparire nell'espressione di un'istruzione di assegnamento
  - può comparire nelle espressioni condizionali all'interno di istruzioni condizionali e iterative

- Gli argomenti di una chiamata di funzione sono a loro volta **espressioni**
  - possono quindi essere espressioni Python qualsiasi purché producano un valore di un tipo previsto dalla funzione (in caso contrario si potrebbe ottenere un messaggio
  - possono altre chiamate di funzione (chiamate *nidificate*).

Librerie di funzioni predefinite

- Molte funzioni predefinite fanno parte di specifiche librerie Python, che a loro volta sono identificate univocamente da un nome simbolico.
- Le funzioni matematiche fanno parte di una libreria di nome **math** e sono descritte in dettaglio nel sito https://docs.python.org/3/library/math.html.
- Alcune di queste sono:
  - `ceil(numero)` # restituisce il più piccolo intero maggiore o uguale a numero (arrotonda per eccesso)
  - `floor(numero)` #restituisce il più grande intero minore o uguale a numero (arrotonda per difetto)
  - `pow(numero1, numero2)` # restituisce la potenza di numero1 elevato numero2
  - `sqrt(numero)` #restituisce la radice quadrata di numero

Librerie di funzioni predefinite

- Oltre alle varie funzioni le librerie presentano anche le definizioni di variabili.
- Ad esempio nella libreria math sono definite delle variabili che contengono il valore (approssimato) delle costanti matematiche
  - `pi` (3,14. . .)
  - `e` (la base dei logaritmi naturali: 2,71. . .)
- Dato che si tratta di variabili (come suggerisce il nome) il loro valore può essere modificato con istruzioni di assegnamento, anche se ciò non è consigliabile (PERCHE' FARLO!!).
- Ovviamente tale modifica è 'visibile' solamente all'interno del programma o blocco del programma in cui tale modificata viene eseguita, ma non nella libreria

Librerie di funzioni predefinite

- La libreria di **random** comprende varie funzioni per la generazione di numeri casuali e sono descritte in dettaglio nel sito https://docs.python.org/3/library/random.html
- Ogni chiamata di tali funzioni produce un numero **pseudocasuale**, indipendente (in teoria) dai valori prodotti dalle chiamate precedenti.
- Alcune di queste sono:
  - `random()` genera un numero reale nell'intervallo [0, 1) da una distribuzione di probabilità **uniforme** in cui ogni valore di tale intervallo ha la stessa probabilità di essere "estratto"
  - `uniform(a,b)` deriva dalla funzione random, alla quale possiamo passare come argomenti i valori dell'intervallo dal quale estrarrà un qualsiasi numero
  - `randint(a,b)` anch'essa deriva dalla funzione random, alla quale possiamo passare come argomenti solamente dei valori interi per definire l'intervallo dal quale estrarrà un numero intero

Librerie di funzioni predefinite

- **Prima** di chiamare una funzione di libreria o delle costanti matematiche è necessaria l'istruzione per importare la libreria in cui è definita la funzione o costante di nostro interesse, altrimenti ogni chiamata a tale funzione restituirà un errore
- **Sintassi**:
- `from nome-libreria import nome-fun/const`
  - nome-libreria è il nome della libreria in cui è definita la funzione
  - nome-fun/const può essere
    - il nome di una specifica funzione o costante di tale libreria (questo consentirà di usare solo tale funzione o costante)
    - il simbolo * indicante **tutte** le funzioni e costanti di tale libreria

Librerie di funzioni predefinite

- Ciononostante l'uso del simbolo * è considerata una cattiva abitudine
  - bassa leggibilità del codice
  - non è possibile stabilire con certezza da quale libreria è stata importata una determinata funzione
  - possibili conflitti con funzioni o costanti di altre librerie o definite dall'utente
  - possibilità concreta di nascondere dei bug
- Preferibile importare le singole funzioni e rinominarle
  - `from nome-libreria import nome-fun/const as nuovo-nome`
- Oppure importare l'intera libreria ed effettuare le chiamate di funzione come segue
  - `import math`
  - `math.sqrt(numero)`

Funzioni definite dall'utente

- Tutti i linguaggi di programmazione consentono inoltre ai programmatori di definire **nuove** funzioni che sono sono dette definite dall'utente (*user-defined*) e permettono di avere numerosi vantaggi:
  - consentono di semplificare la scrittura di programmi complessi suddividendoli in più parti (funzioni), ciascuna delle quali svolge un compito **distinto** dalle altre, e può essere sviluppata in modo **indipendente** da esse
  - l'esecuzione di una funzione può essere richiesta in diversi punti di uno stesso programma, senza dover scrivere più volte le sue istruzioni
  - una stessa funzione può essere usata in programmi diversi

Funzioni definite dall'utente

- La sintassi della chiamata di tali funzioni è identica a quella delle funzioni predefinite.
- La definizione di una nuova funzione avviene attraverso l'istruzione **def** e si compone di:
  - **instestazione** della funzione (la prima riga contenente il nome della funzione e dei parametri)
  - **corpo della fuzione** che contiene la sequenza di istruzioni che dovrà eseguire

- **Sintassi**:
- def nome-funzione (par1, . . . , par*n*) :

  corpo della funzione
  - nome-funzione è un nome simbolico scelto dal programmatore, con gli stessi vincoli a cui sono soggetti i nomi delle variabili
  - par1, . . . , par*n* sono nomi (scelti dal programmatore) di variabili, dette parametri della funzione, alle quali l'interprete assegnerà i valori degli **argomenti** che verranno indicati nella **chiamata** della funzione
  - corpo della funzione è una sequenza di una o più istruzioni **qualsiasi**, ciascuna scritta in una riga **distinta**, con un **rientro** di almeno un carattere, identico per **tutte** le istruzioni

Funzioni definite dall'utente

- Tipicamente il corpo della funzione si conclude con l'istruzione return, che indica anche il valore che la funzione dovrà **restituire** come risultato della sua chiamata
- **Sintassi**: return espressione
  - dove espressione è un'espressione Python **qualsiasi**.
- Questa istruzione può essere usata solo **solo** all'interno di una funzione.
- Se una funzione **non** deve restituire nessun valore:
  - l'istruzione return può essere usata, senza l'indicazione di nessuna espressione, per concludere l'esecuzione della funzione
  - se non si usa return, l'esecuzione della funzione terminerà dopo l'esecuzione dell'ultima istruzione del suo corpo

Funzioni definite dall'utente: chiamata

- L'esecuzione di un programma che contiene la definizione di una funzione **non** comporta l'esecuzione delle istruzioni della funzione: tali istruzioni verranno eseguite solo attraverso una **chiamata** della funzione.

- Come per le funzioni predefinite, anche per quelle definite dall'utente il numero di argomenti indicati nella chiamata dovrà corrispondere al numero di argomenti previsti nella definizione, cioè al numero dei parametri indicati nell'intestazione.

- L'interprete esegue la chiamata di una funzione nel modo seguente:
  - **copia** il valore di ciascun argomento nel parametro corrispondente (quindi tali variabili possiedono già un valore nel momento in cui inizia l'esecuzione della funzione)
  - esegue le istruzioni del corpo della funzione, fino a incontrare l'istruzione return oppure l'ultima istruzione del corpo
  - se l'eventuale istruzione return è seguita da un'espressione, restituisce il valore di tale espressione come risultato della chiamata

Definizione di nuove funzioni: esempio

- Volendo definire una funzione che restituisce il più grande tra due numeri ricevuti in input. Scegliamo **massimo** come nome della funzione, e **a** e **b** come nomi dei suoi parametri. La funzione può essere definita come segue:

```
def massimo (a, b) :
    if a > b :
        return a
    else :
        return b
```

- Le variabili (parametri) a e b della funzione massimo **avranno già un valore** nel momento in cui inizierà l'esecuzione delle istruzioni del corpo della funzione, però non devono essere definiti per mezzo di istruzioni di assegnamento nel **corpo** della funzione, ma vengono definiti nella **chiamata** della stessa funzione.

Funzioni senza argomenti o risultato

- Come caso particolare è possibile definire funzioni che non ricevono argomenti oppure funzioni che non restituiscono un risultato.
- Se si vuole definire una funzione che **non riceve argomenti**, nell'intestazione si dovranno solo scrivere le parentesi tonde, senza il nome di alcun parametro al loro interno. In modo analogo, ci si comporterà nella chiamata alla funzione
- Se si vuole definire una funzione che **non restituisce** un risultato, non si dovrà usare nel suo corpo nessuna istruzione return

```
def saluto() :
    print (Buongiorno)
```

- Cosa succede se effettuo delle chiamata di funzione `s = saluto()` oppure `saluto('ciao')` ?

Esercizi

6. Definire una funzione che riceva come argomento un numero intero non negativo n e ne restituisca il fattoriale

7. Definire una funzione che riceva come argomento un numero intero non negativo n e restituisca la somma dei primi n termini della serie armonica