Laurea in Ingegneria Gestionale

Corso di Fondamenti di Informatica
A.A. 2017/2018

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Iterazioni 2

Le seguenti slide sono tratte principalmente dal lavoro di Giorgio Fumera (Univ. degli Studi di Cagliari) e riadattate da Raffele Nicolussi (Sapienza) e Domenico Lembo (Sapienza). Alcune sono invece una rielaborazione di slide analoghe di Diego Calvanese (Univ. di Bolzano).

Esempi di programmi Python che usano cicli: Massimo Comun Divisore (MCD)

Esempio: Leggere due interi positivi e calcolare il **massimo comun divisore**.

Es.:
MCD (12, 8) = 4
MCD (12, 6) = 6
MCD (12, 7) = 1

MCD: algoritmo 1

1) Sfruttando direttamente la definizione di MCD

- osservazione: 1 ≤ MCD(m, n) ≤ min(m, n)
  ⇒ si provano i numeri compresi tra 1 e min(m, n)
- conviene iniziare da min (m, n) e scendere verso 1

algoritmo *stampa massimo MCD di due interi positivi letti da tastiera*
leggi *m* ed *n*
inizializza *mcd* al minimo tra *m* ed *n*
while non si è trovato un divisore comune
    if *mcd* divide sia *m* che *n*
        si è trovato un divisore comune
    else decrementa *mcd* di 1
stampa *mcd*

Osservazioni su algoritmo 1 per MCD

Osservazioni:

- il ciclo termina sempre perchè ad ogni iterazione
  - o si è trovato un divisore
  - o si decrementa *mcd* di 1 (al più si arriva ad 1)
- per verificare se si è trovato il MCD si utilizza una variabile booleana (usata nella condizione del ciclo)

Esempi di programmi Python: MCD v1

```
# File MCD_1.py

m = int(input("Primo numero: "))
n = int(input("Secondo numero: "))
if m < n :
    mcd = m
else :
    mcd = n
trovato = False
while not trovato:
    if m % mcd == 0 and n % mcd == 0  :
        trovato = True
    else :
        mcd = mcd - 1
print ("Il MCD di", m, "e", n, "e'", mcd)
```

Esempi di programmi Python: MCD v2

Una variante più sintetica (e più elegante) dello stesso programma (disponibile nel *file* `MCD_2.py`): l'istruzione iterativa ha il solo scopo di decrementare il valore della variabile `mcd` fino a che questo **non** è un divisore comune di *a* e *b*.

```
m = int(input("Primo numero: "))
n = int(input("Secondo numero: "))
if m < n :
    mcd = m
else :
    mcd = n
while not (m % mcd == 0 and n % mcd == 0)  :
    mcd = mcd - 1
print ("Il MCD di", m, "e", n, "e'", mcd)
```

Quante volte viene eseguito il ciclo?

caso migliore: 1 volta (quando m divide n o viceversa)
p.es. MCD (500, 1000)

caso peggiore: min (m, n) volte (quando MCD (m, n) = 1)
p.es. MCD (500, 1001)

⇒ algoritmo si comporta male se m e n sono grandi e MCD(m,n) è piccolo

MCD: algoritmo 2 (Euclide)

2) Permette di ridursi più velocemente a numeri più piccoli, sfruttando la seguente proprietà:

$$MCD(m,n) = \begin{cases} m \quad (\text{o } n), & \text{se } m = n \\ MCD(m-n,n), & \text{se } m > n \\ MCD(m,n-m), & \text{se } m < n \end{cases}$$

Es.: MCD(12, 8) = MCD(12−8, 8) = MCD(4, 8−4) = 4

Come si ottiene un algoritmo?
Si applica ripetutamente il procedimento fino a che non si ottiene che m = n.

algoritmo *di Euclide per il MCD di due interi positivi*
leggi *m* ed *n*
while *m* != *n*
  sostituisci il maggiore tra *m* ed *n* con la differenza tra
    il maggiore ed il minore
stampa *m* (oppure *n*)

Esempi di programmi Python MCD v3

```
# File MCD_3.py

m = int(input("Primo numero: "))
n = int(input("Secondo numero: "))
while m != n :
    if m > n :
        m = m - n
    else :
        n = n - m
print ("Il MCD e'", m)
```

Cosa succede se m=n=0? ⇒ il risultato è 0
E se m=0 e n!=0 (o viceversa)? ⇒ si entra in **un ciclo infinito**

Se si vuole tenere conto del fatto che l'utente possa immettere una qualsiasi coppia di interi, è necessario inserire una verifica sui dati in ingresso.

Esercizio: stampa sequenza numeri pari

Scrivere un programma che stampi tutti i numeri pari compresi tra 1 e un numero acquisito attraverso la tastiera

Stampa sequenza numeri pari -- Soluzione

```
# File stampaNumeriPari.py

massimo = int(input("stampa numeri pari da 1 a?: "))
n = 2
print ("I numeri pari tra 1 e", massimo, " sono:")
while n <= massimo :
    print (n)
    n = n + 2
```

Esercizio: somma i primi m termini della serie armonica

Scrivere un programma che **calcola la somma dei primi *m* termini della serie armonica**, per un dato valore di *m*:

$$\sum_{k=1}^{m} \frac{1}{k}$$

Ad esempio, se in input viene inserito 5 (quindi m = 5), il risultato sarà 2.283333333333333 (dato da 1+1/2+1/3+1/4+1/5)

Somma i primi m termini della serie armonica -- soluzione

```
# File serieArmonica.py

m = int(input("Numero di termini della serie armonica: "))
serie = 0
k = 1
while k <= m :
    serie = serie + 1 / k
    k = k + 1
print ("Somma dei primi", m, " termini: ", serie)
```

Esercizio: calcolo del fattoriale

Scrivere un programma che calcola il fattoriale di un dato numero naturale *n*, definito come segue:

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

Calcolo del fattoriale -- soluzione

Il procedimento di calcolo è analogo a quello per la somma di una sequenza di numeri: il risultato viene calcolato memorizzando nella variabile `fatt` il valore del prodotto **parziale** dei numeri 1, 2, . . . , *n*, che vengono scanditi per mezzo di un'istruzione iterativa.

```
# File fattoriale.py

n = int(input("Inserire un numero naturale non negativo: "))
fatt = 1
k = 2
while k <= n :
    fatt = fatt * k
    k = k + 1
print ("Il fattoriale di", n, "e'", fatt)
```

Calcolo del fattoriale -- soluzione

Soluzione che non usa un contatore addizionale per controllare il ciclo, ma sfrutta il numero in input ("contando all'inverso").

```
# File fattoriale_2.py

n = int(input("Inserire un numero naturale non negativo: "))
print ("Il fattoriale di", n, "e'", end = "")
fatt = 1
while n > 1 :
    fatt = fatt * n
    n = n - 1
print (fatt)
```

Nota: `end = ""` nella `print` consente di non andare a capo. Di fatto sovrascrive lo `\n` che di default la `print` aggiunge alla fine di ciascuna stringa

Esercizio: conversione binaria

Scrivere un programma che converte un numero naturale (in base 10), nella sua rappresentazione iin base due. usando **il noto procedimento delle divisioni successive per due.**

Ad esempio, per la conversione del numero 13 in binario abbiamo

Divisione intera per 2	**risultato**	**resto**
13 // 2	6	**1**
6 // 2	3	**0**
3 // 2	1	**1**
1 // 2	0	**1**

Il risultato della conversione è **1101,** cioè $(13)_{10} = (1101)_2$

Conversione binaria -- soluzione

Nel programma seguente, notate che le cifre vengono vengono calcolate dalla meno significativa alla più significativa, pertanto la concatenazione nell' "accumulatore" ris avviene concatenando la cifra calcolata in testa alla stringa

```
# File conversioneBaseDue.py

n = int(input("Numero naturale da convertire in base due: "))
ris=""
if n == 0 :
    print (0)
while n != 0 :
    ris = str(n % 2) + ris
    n = n // 2
print("la rappresentazione in base 2 e':",ris)
```

Esercizio: verifica numero primo

Scrivere un programma che verifica se un dato numero naturale *N* letto da `input` sia primo o meno (un numero primo (in breve è un numero intero positivo che abbia esattamente due divisori; la sequenza dei numeri prima incomincia pertanto da 2).

Suggerimento: per verificare se un numero è primo, scandite (per mezzo di un'istruzione iterativa) tutti i possibili divisori di *N* (diversi da 1), cioè i valori compresi tra 2 e $\lfloor N/2 \rfloor$ (estremi inclusi, dove $\lfloor x \rfloor$ indica la parte intera di *x*). Appena trovate un divisore potete uscire dal ciclo e stampare il risultato.

verifica numero primo – soluzione

```
#File verificaNumeroPrimo.py

n = int(input("Inserire un numero naturale maggiore di 1: "))
primo = True
divisore = 2
while divisore <= n / 2 and primo == True :
    if n % divisore == 0 :
        primo = False
    else :
        divisore = divisore + 1
if primo == True :
    print ("Il numero", n, " e' primo")
else:
    print ("Il numero", n, " non e' primo")
```

verifica numero primo – soluzione (commenti)

Il procedimento codificato tiene traccia per mezzo della variabile `primo` del fatto che sia stato trovato o meno un divisore.

A tale variabile si assegna inizialmente il valore `True` (non è stato ancora trovato nessun divisore, quindi il numero viene considerato primo). Durante l'iterazione le si assegnerà il valore `False` se verrà trovato un divisore, e in questo caso l'iterazione potrà terminare senza completare la scansione dei possibili divisori. Il risultato viene mostrato per mezzo di un messaggio che dipende dal valore della variabile `primo`.

Nel ciclo, la variabile `primo` è usata in modo analogo a quanto fatto con la variabile `trovato` nel programma MCD v1

Esercizio: stampa numeri primi

Scrivere un programma che stampa tutti i numeri primi compresi tra 1 e un dato numero naturale positivo letto da input

stampa numeri primi -- soluzione

```
# File sequenzaNumeriPrimi.py

n = int(input("Inserire un numero naturale maggiore di 1: "))
print ("I numeri primi tra 1 e", n, "sono:")
cont = 2
while cont <= n :
    primo = True
    divisore = 2
    while divisore <= cont / 2 and primo == True :
        if cont % divisore == 0 :
            primo = False
        else :
            divisore = divisore + 1
    if primo == True :
        print (cont)
    cont = cont + 1
```

stampa numeri primi – soluzione (commenti)

Si noti che nel programma precedente sono usate due istruzioni iterative nidificate: la prima scandisce tutti i numeri dell'intervallo da considerare, la seconda (nidificata) **verifica se il numero in esame sia** primo (sfruttando il programma dell'esercizio precedente).

Esercizio: calcolo polinomio

Scrivere un programma che calcoli il valore di un polinomio di grado qualsiasi, $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$, per un dato valore della variabile x. I dati d'ingresso sono il grado del polinomio, il valore di x e i valori dei coefficienti

Suggerimento: scrivere un ciclo che chiede un coefficiente alla volta a partire dal termine di grado **inferiore** e calcola il risultato (parziale) del valore del polinomio, accumulando il risultato in una variabile che funge da accumulatore.

calcolo polinomio -- soluzione

```
# File polinomio.py

n = int(input("Grado del polinomio: "))
x = int(input("Valore della variabile: "))
polinomio = 0
potenza_x = 1
k = 0
while k <= n :
    print ("Inserire il coefficiente di grado ", k)
    a = int(input())
    polinomio = polinomio + a * potenza_x
    potenza_x = potenza_x * x
    k = k + 1
print ("Il valore del polinomio e'", polinomio)
```

Ulteriori Esercizi

Scrivere programmi Python che realizzino le seguenti operazioni e stampino i risultati nella finestra della *shell* :

- Calcolare il valore più grande in una sequenza di numeri qualsiasi (dati d'ingresso: la lunghezza della sequenza e i suoi elementi)
- Stampare tutti i divisori di un dato numero naturale
- Calcolare il minimo comune multiplo (MCM) di due numeri naturali (dati d'ingresso: i valori dei due numeri).
  Suggerimento: il MCM tra due numeri *a* e *b* è definito come il più piccolo numero naturale del quale sia *a* che *b* siano divisori...

Le soluzioni sono rispettivamente nei file `massimo.py`, `divisori.py`, `mcm.py`

L'istruzione `break`

Si è visto che l'esecuzione di 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.

L'istruzione `break`: esempio numero primo

Un buon esempio di uso dell'istruzione break è la codifica dell'algoritmo visto in precedenza per determinare se un numero naturale sia primo (si veda `verificaNumeroPrimo.py`).

Nella versione seguente (disponibile nel *file* `verificaNumeroPrimo_2.py`) l'iterazione viene conclusa con `break` non appena viene trovato un divisore. In questo modo non è necessario aggiungere la condizione `primo == True` nell'espressione condizionale dell'istruzione `while`.

Analogamente possiamo usare l'istruzione `break` per evitare di usare la variabile booleana `trovato` nel programma MCD v1.

L'istruzione break: esempio numero primo

```
n = int(input("Inserire un numero naturale: "))
primo = True
divisore = 2
while divisore <= n / 2 :
    if n % divisore == 0 :
        primo = False
        break
    else :
        divisore = divisore + 1
if primo == True :
    print ("Il numero ", n, "e' primo")
else:
    print ("Il numero ", n, "non e' primo")
```

L'istruzione break: esempio MCD (variante di algoritmo 1 per MCD)

```
# File MCD_4.py

a = int(input("Primo numero: "))
b = int(input("Secondo numero: "))
if a < b :
  mcd = a
else :
  mcd = b
while True:
    if a % mcd == 0 and b % mcd == 0  :
       break
    else :
       mcd = mcd - 1
print ("Il MCD di", a, "e", b, "e'", mcd)
```

L'istruzione `break`: esercizi

Riscrivere il programma per il calcolo del MCM di due numeri naturali usando l'istruzione `break`.