Riassunto di Programmazione 1 - Python

# Python

Python è un linguaggio di programmazione orientato agli oggetti che fa della semplicità, dinamicità e flessibilità i suoi principali vantaggi, tuttavia è comunque meno performante rispetto ad altri linguaggi come il C che, pur essendo più complesso, è più performante rispetto al primo. Un programma scritto col Python deve essere salvato con l’istruzione “.py”.

## Il primo programma in Python

La versione utilizzata in questo programma e nei successivi è Python 3, non compatibile con la versione precedente a causa della differente scrittura delle istruzioni.

print(“Hello World”)

L’esempio sopra permette di stampare a video la scritta Hello World. Ci sono anche istruzione che permettono di acquisire numeri, caratteri e stringhe e successivamente stamparle:

nome=input(“Come ti chiami?)

print=(“Ciao”, nome)

nome è una variabile, una “scatola” dove è possibile metterci dei dati all’interno.

In questo caso il programma stampa la stringa “Come ti chiami”, attende un input da tastiera ed infine stampa “Ciao” seguito dal contenuto della variabile nome.

I dati presi con l’istruzione input vengono considerate come stringhe, è possibile convertire le stringhe di numeri in interi attraverso l’istruzione int():

int(“55”) #il risultato è 55

int(“il numero è 55”) #darà un errore perché sono presenti caratteri che non sono numeri

“#” indica un commento, una riga di codice che viene scartata automaticamente dal compilatore e ed solitamente utilizzata per segnare le istruzioni da non considerare.

Oltre a int(), esistono altre funzioni che permettono di convertire i numeri:

* str(): converte il numero in una stringa;
* float(): converte una stringa o un intero in un numero decimale;

## Le Funzioni

Le funzioni servono per semplificare il programma e scrivere meno righe di codice, le variabili utilizzate dentro una funzione valgono soltanto dentro di essa.

def Saluta():

m=input(“Inserisci un nome”)

print(“Ciao”, m) #prendiamo come esempio Orazio

print(“Letto Tutto”)

def Risaluta():

m=”Clarabella”

Saluta()

Print(“Riciao”, m)

Il risultato è: “Ciao Orazio” e dopo “Riciao Clarabella”. Nonostante m abbia cambiato valore quando viene eseguita la funzione Saluta(), la stringa “Clarabella” viene mantenuta all’interno della funzione di partenza che è Risaluta().

Esistono funzioni che permettono di eseguire le istruzioni al suo interno attraverso uno o più dati in ingresso:

def AreaRet(base, altezza):

area=base\*altezza

return area

L’istruzione return restituisce il valore interessato, in questo caso quello all’interno della variabile area.

L’esempio sopra riporta un caso in qui abbiamo dei valori da dare in ingresso senza l’istruzione input, questo permette di semplificare i programmi e scrivere meno righe di codice.

Come nella funzione Riciao(), è possibile mettere una funzione dentro l’altra:

def AreaRet(base, alt):

area=base\*alt

return area

def AreaTri(base, alt):

arearet=AreaRet(base, alt)

areatri=arearet/2

return areatri

## Il ciclo determinato (For)

Il ciclo for permette di eseguire più volte un certo numero di istruzioni.

Def Somma(n):

print(“Inizio”)

for i in range(1,n+1):

somma=somma+1

print(“Fine”) # se n=5 verrà eseguita la somma 5 volte.

return somma

A differenza degli altri linguaggi, il ciclo for di Python esclude l’ultimo numero. Quindi se vogliamo che il numero n venga considerato, inseriamo n+1 nell’istruzione del ciclo.

È possibile modificare il contatore inserendo un terzo numero nel range del ciclo.

For i in range(1, n+1,2) # i aumenta di due ogni volta fino a quando non raggiunge n.

For i in range(10,1,-1) # i diminuisce di 1 ogni volta fino a quando non raggiunge 1.

L’istruzione break permette di uscire dal ciclo anche se i non è ancora arrivato al massimo:

for i in range(1,10):

if i==5:

break #il programma è uscito dal ciclo ma I non è arrivato al valore 5 e non a 10

L’istruzione continue è molto simile a break con la differenza che fa continuare il ciclo non considerando le istruzioni successive.

Lista=[]

for i in range(1,10):

if i==5:

continue #le istruzione dopo continue non vengono considerate

Lista+=i

Return Lista

#il risultato sarà [1,2,3,4,6,7,8,9]

## Il confronto con if, else ed elif

Le istruzioni if ed else si utilizzano si devono eseguire delle istruzioni quando si verificano determinate condizioni. Se la condizione di if è vera allora le istruzioni vengono eseguite, se è falsa invece queste ultime vengono ignorate e passa alle successive.

def minimo(a,b):

if a<b:

return a

else:

return b

L’istruzione elif si comporta come un’else ma, a differenza di quest’ultimo, ha bisogno di soddisfare delle condizioni per eseguire il codice.

Programma che fa le 4 operazioni:

def calcolatrice(a,b,s)

if s==”+”:

return a+b

elif s==”-“

return a-b

elif s==”\*”

return a\*b

elif s==”//”

return a//b

else: #se la variabile s non soddisfa le condizioni di if ed elif, vengono eseguite

return None #le istruzioni di else

## le stringhe e la funzione len()

La funzione len() permette di trovare il numero di caratteri (spazi compresi) di cui è composta la stringa

len(“ciao”) #il risultato è 4, cioè il numero di caratteri della parola ciao.

len(“ciao, come va?”) #il risultato è 14.

Una stringa si comporta come un vettore:

stringa=”ciao”

stringa[0]=”c”

stringa[1]=”i” #se gli indici sono negativi, i caratteri della stringa vengono stampati

stringa[2]=”a” # al contrario.

stringa[3]=”o”

## L’istruzione assert

L’istruzione assert permette di bloccare un programma quando non vengono rispettate determinate condizioni:

def divisione(a,b):

assert b!=0, “Errore” #se b è uguale a 0, il programma si blocca e stampa un messaggio di errore.

Return a//b

Se combinato con l’istruzione isinstance, assert può accettare valori solo del tipo specificato:

assert isinstance(b,int), “errore” #accetta solo valori interi

assert isinstance(b,str), “errore” #accetta solo stringhe

## Le Liste

Le liste in python sono comparabili agli array negli altri linguaggi come C, Java, ecc. e sono variabili che possono contenere più valori al loro interno (anche di più tipologie), si indicano con le parentesi quadre.

Lista0 = [] #questa lista è vuota

Lista1 = [ 1, 2, 3, 4 ] #lista formata solo da numeri interi

Lista2 = [ “ciao”, “casa”, “montagna”, “mare”] #lista formata solo da stringhe

Lista3 = [ 1, “ciao”, 2.5, “c”, 22, “425”] #lista formata da più tipologie di valori

Per aggiungere un valore all’interno di una lista, bisogna inserire l’elemento da aggiungere nelle parentesi quadre. Per cambiare un valore, invece, lo si può sovrascrivere come una normalissima variabile attraverso l’utilizzo dell’indice che ne indica la posizione (gli indici partono da 0)

Lista1 += [5] # ora la lista1 contiene anche il valore 5.

Lista1[2] = 7 #il valore 3 è stato sostituito da 7 nella lista1, ora è 1,2,7,4

Come nel caso delle stringhe, anche le liste si possono sommare tra loro.

Lista4=Lista1+Lista2 #la lista4 conterrà sia i valori di lista1 che di lista2

È anche possibile utilizzare la funzione len per ricavare il numero di valori contenuti all’interno.

N = len(Lista1) #il valore di N è 4, cioè il numero di valori contenuti in Lista1

Per sapere se un valore si trova all’interno di una lista o meno, si utilizza l’istruzione in che, se il valore interessato risulta presente all’interno, restituirà un valore booleano (True o False)

Control1 = 4 in Lista1 #la variabile Control1 avrà True come valore perché 4 è contenuto in Lista1

Control2 =8 in Lista1 #la variabile Control2 avrà False come valore perché 8 non è contenuto in Lista1

Utilizzando l’istruzione .split() è possibile generare una lista con tutte le sottostringhe all’interno:

stringa.split() #divide le parole separate da uno spazio

stringa.split(“-“) #divide le parole separate da “-“

Ultimo ma non meno importante è l’ordinamento delle suddette liste ed è possibile scriverlo in due differenti modi:

* sorted(lista) crea una lista ordinato a seconda dei parametri dati;
* Lista.sort() ordina la lista interessata

## Scorrere gli elementi di una lista

Un modo più semplice per far scorrere una lista è utilizzare una versione modificata del classico ciclo for:

lista=[1,2,3,4,5,6,7]

ris=[]

for a in lista: #a prende il valore di lista[i], cioè quello che ha in questo momento.

ris+=[a,a]

Return ris

A differenza del for normale, questo tipo di ciclo non è facilmente invertibile, per questo è utilizzato principalmente per applicazioni semplici o per lo scorrimento di liste.

Un’errore comune è il seguente:

for i in cont:

cont[a]=2 #Sbagliato, a non può essere l’indice della lista da scorrere.

lett[a]=2 #Giusta, a può essere l’indice di questa lista in quanto non è quella da scorrere.

In poche parole a non può essere l’indice della lista da scorrere ma può esserlo in tutte le altre.

## Il ciclo indeterminato (While)

Il ciclo while permette di eseguire un certo numero di istruzioni infinite volte fino a quando le condizioni non vengono più rispettate.

Def ciclowhile(n):

c=true

while c==true:

if n%3==0:

c=false

return n

## Accesso a file esterni al programma

Il linguaggio Python di visualizzare file esterni attraverso il ciclo for spiegato precedentemente ed a determinate istruzioni:

def visualizza(nome):

assert isinstance(nome,str),”non è una stringa”

f=open(nome) #se il file esiste, viene aperto ed il suo contenuto viene messo nella variabile f.

for s in f:

print(s,end=””)

close(f) # chiude il file

return

l’istruzione print stampa automaticamente una linea vuota dopo la stringa, svuotando la variabile end (che di default vale \n, cioè il ritorno a capo) si otterrà una lettura molto più simile all’originale.

## Scrittura di un file di testo

Oltre alla lettura di file di testo, Python permette anche di creare nuovi file di testo e scriverci all’interno:

def scrittura(lista):

assert isinstance(lista,list),”non è una lista”

f=open(“lista.txt”,”w”) #creo un file di nome lista.txt dove inserirò i valori all’interno della lista

for s in lista:

f.write(s) #scrivo il valore s nel file di testo

close(f) #chiude il file

return

## I Dizionari

I dizionari sono tipi di variabili molto simili alle liste, con la differenza che hanno come indice una stringa al posto di un numero.

def contaparole(stringa):

lista=stringa.split() #creo una lista formata da tutte le parole separate da uno spazio

dizio={} #dichiaro un dizionario vuoto

for p in lista: #scorro gli elementi della lista creata in precedenza

if p in dizio: #se la parola è presente nel dizionario, incremento di uno

dizio[p]+=1

else: #se la parola non è presente, la aggiungo nel dizionario

dizio[p]=1

return dizio

per inserire gli elementi di un dizionario all’interno di una lista, si usa l’istruzione “list(dizio.items()”, in questo modo creiamo una lista forma da tuple, cioè una sottolista della lista principale.

## Input da riga di comando

I programmi scritti in python permettono, attraverso determinate istruzioni, di prendere valori dalla Command Line come in questo esempio:

#! /usr/bin/env python 3

#questo è un commento speciale che permette di eseguire i programmi python senza l’istruzione pyhon3

Import sys #sys è una libreria con le istruzioni necessarie per la lettura da command line

#PARTE DEDICATA ALLE FUNZIONI

Def fattoriale(n):

fatt=1

for i in range(1,n+1):

fatt\*=i

return fatt

#PROGRAMMA PRINCIPALE

If len(sys.argv)!=2:

print(“inserire solo due valori”)

else:

print(fatt(int(sys.argv[1]))) #l’input, che è una stringa, viene convertita in intero per evitare errori