lezione03

September 30, 2025

1 Fondamenti di programmazione

2 Canale A-L - Prof. Sterbini

3 AA 25-26

4 Lezione 3

5 RECAP lezione 2

- stringhe, stringhe multilinea, interpolazione
- variabili, riferimenti e namespace
- assegnamento
- conversione stringhe <-> numeri
- confronti

6 Operatori di assegnamento 'potenziato' precedendo = con l'operatore

Può far comodo modificare una variabile aggiungendole, sottraendole, moltiplicandola o dividendola per un valore - `A += 1` **incremento** (come `A = A+1` ma più compatto) - `A -= 2` **decremento** (come `A = A-2` ma più compatto) - `A *= 3` **moltiplicazione** (come `A = A*3` ma più compatto) - `A /= 4` **divisione** (come `A = A/4` ma più compatto)

[166]:
```
# Esempio
A = 42
print(A)
A += 1
print('più    1', A)
A -= 12
print('meno  12', A)
A *= 10
print('per   10',A)
A //= 5
print('diviso 5',A)
A %= 7
print('il resto per 7',A)
A **= 2
print('elevato alla 2',A)
B ='Paperino '
B += 'e Minnie'
print(B)
```

```
42
più    1 43
meno  12 31
per   10 310
diviso 5 62
il resto per 7 6
elevato alla 2 36
Paperino e Minnie
```

[158]:
```
# Se applicati alle stringhe, gli assegnamenti potenziati '+=' e '*='
# eseguono concatenazioni o ripetizioni
A = 'Paperoga '
print(A)
A += 'e Gastone | '
print(A)
A *= 3
print(A)
```

```
Paperoga
Paperoga e Gastone |
Paperoga e Gastone | Paperoga e Gastone | Paperoga e Gastone |
```

7 Assegnamenti multipli

In Python possiamo assegnare contemporaneamente più variabili in una sola istruzione "'python A, B, C = 1, 2, 3

A contiene 1 B contiene 2 C contiene 3

Basta che il numero di espressioni calcolate a destra sia uguale al numero di variabili a sinistra

A destra può esserci un qualsiasi contenitore list/tuple/dict/set (vedi dopo)

8

[162]:
```
# Esempio
L = 'Paperino e Minnie'.split()    # produce la lista ['Paperino', 'e',␣
 ↪'Minnie']
P, e, M = L                        # assegno i 3 elementi della lista a 3␣
 ↪variabili
print(P)                           # e le stampo
print(e)
print(M)
```

```
Paperino
e
Minnie
```

8.1 Sono ESTREMAMENTE COMODI per

- estrarre informazioni organizzate in una lista o tupla per **posizione**
- scambiare tra loro i valori di due o più variabili
- spacchettare i valori di ritorno di una funzione (lo vediamo dopo)

[208]:
```
# ESEMPIO: spacchettiamo una lista che contiene i dati di una persona,␣
 ↪organizzati per posizione
#          nome      cognome    età altezza figli
SCHEDA = ['Andrea', 'Sterbini', 64, 186,    2]
nome, cognome, eta, altezza, figli = SCHEDA
print('nome\t',nome)
print('cognome\t',cognome)
print('età\t',eta)
print('altezza\t',altezza)
print('figli\t',figli)
```

```
nome     Andrea
cognome  Sterbini
età      64
altezza  186
figli    2
```

[168]:
```
# ESEMPIO: scambiamo due variabili
prima = 'Paperino'
seconda = 'Minnie'
prima, seconda = seconda, prima  # scambio i valori
print(prima)
print(seconda)
```

```
Minnie
Paperino
```

8.2 Come mai lo scambio funziona?

Se corrispondesse alla sequenza di istruzioni

```
prima = seconda
seconda = prima
```

otterremmo che in entrambe le variabili c'è 'Minnie'

[209]:
```
prima = 'Paperino'
seconda = 'Minnie'
prima = seconda
seconda = prima
prima, seconda
```

[209]:
```
('Minnie', 'Minnie')
```

Invece quello che fa Python è: - PRIMA calcola l'espressione a destra della **'='** ovvero **seconda, prima** che produce la coppia **('Minnie', 'Paperino')** - poi assegna a ciascuna delle due variabili a sinistra dell' **'='** i due valori della coppia

Dato che i due valori sono stati entrambi già letti dalle variabili, non si perdono e possono essere assegnati senza problemi

[215]:
```
# Esempio contorto
A, B, C = 'uno', 'due', 'tre'

B, A, C, C, B, A = A, A, A, A, C, B       # solo gli ultimi 3 assegnamenti␣
 ↪hanno significato :-D
A, B, C                                   # non ci siamo persi nulla!
```

[215]:
```
('due', 'tre', 'uno')
```

8.3 Condizioni e percorsi alternativi

Per scegliere di eseguire parti diverse di codice in situazioni diverse si usa l'istruzione **if** e delle condizioni (booleane) che valgono **True** o **False**

Sintassi:

```
if condizione1:
    istruzioni eseguite se condizione1 è True
```

```
elif condizione2:          # sezione opzionale (e ripetibile + volte)
    istruzioni eseguite
    se condizione1 è False e condizione2 è True
else:                      # sezione opzionale
    istruzioni eseguite
    se sia condizione1 che condizione2 sono False
```

NOTATE CHE: il codice è organizzato in modo particolare, i blocchi di istruzioni sono **indentati** (spostati a destra) rispetto alla parola chiave che li introduce (if, elif, else)

8.4 Indentazione .... ma come mai?

In C/C++/Java usiamo le parentesi graffe `{ }` per raggruppare i blocchi di codice e i punto-e-virgola `;` per separare le istruzioni

Questo rende possibile scrivere codice molto disordinato e poco leggibile come questo su una sola riga

```
if(condizione1){istruzione1;istruzione2;istruzione3;}else{istruzione4;istruzione5;istruzione6;}
```

Negli anni si è diffuso uno stile di scrittura ordinato e più leggibile, come questo:

```
if (condizione1) {  // con le strutture di controllo ben visibili
    istruzione1;    // con una istruzione per ciascuna riga
    istruzione2;    // commentata se serve, sulla stessa riga
} else {
    istruzione3;    // e ciascun blocco logico di istruzioni
    istruzione4;    // spostato un po' a destra per evidenziarlo
}
```

Questo stile ***indentato*** è molto leggibile e fa parte delle regole obbligatorie di scrittura del Python

E visto che ogni istruzione è su una riga diversa e i blocchi sono indentati, non c'è più bisogno di parentesi graffe e punto-e-virgola

9 Altrimenti possiamo usare l'istruzione `match` (simile allo switch/case di C/C++/Java)

Serve a confrontare un valore con più valori possibili, **esaminandone anche la struttura!**

```
match variabile:
    case valore1:              # match esatto
        istruzioni se variabile == valore1
    case valore2 | valore3:    # match con più alternative
        istruzioni se variabile == valore2 OPPURE valore3
    case [x, y]:               # match strutturale
        istruzioni se variabile è una lista di 2 valori qualsiasi
        i valori x ed y vengono spacchettati e possono essere usati in questo blocco
    case _:                    # caso di default
        istruzioni se variabile non è uguale a nessuno dei valori precedenti
```

NOTA: diversamente da C/C++/Java, non c'è bisogno di `break` alla fine di ciascun blocco perchè i diversi blocchi sono mutualmente esclusivi (ne viene eseguito solo uno oppure nessuno)

FDPLEZ3

9.1 Cosa stampa il programma ?

[174]:
```
# soluzione
is_raining, got_umbrella, have_money   = True, False, True
if not is_raining:                                          # False
    print('go out')
elif not got_umbrella and have_money:                       # True
    print('buy umbrella and go out')
elif got_umbrella:                                          # non viene␣
 ↪esaminato
    print('go out since got umbrella')
else:                                                       # non eseguito
```

```
    print('stay home')
```

```
buy umbrella and go out
```

[176]:
```
# soluzione
X = "Paperino ama Paperina".split()
match X:
    case 42:
        print("La risposta alla domanda fondamentale sulla vita, l'universo e␣
 ↪tutto quanto")
    case "Paperino ama Paperina":
        print("Ma anche Gastone?")
    case "Paperino" | "ama" | "Paperina":
        print("E Topolino ama Minnie")
    case [ soggetto, verbo, oggetto ]:
        print( f"{oggetto} è {verbo}to/a da {soggetto}" )
    case _:
        print( "Non ho capito cosa hai detto" )
```

```
Paperina è amato/a da Paperino
```

10 Cicli ed iterazione/ripetizione

10.1 Caso 1: quando *SAPETE QUANTE* iterazioni/ripetizioni dovete fare ALLORA è meglio usare il FOR

Sintassi:

```
# viene definita la variabile 'elemento' e gli viene
# via via assegnato il prox valore della sequenza
for elemento in sequenza:
    blocco di codice da ripetere
    per ciascun elemento della sequenza
else:        # opzionale
    blocco che viene eseguito se si è completato il
    ciclo normalmente senza interruzioni (break) o errori
```

[2]:
```
# ESEMPIO
for X in [1, 2, 3, 4]:  # ripeto 4 volte, con X che prende i valori 1, 2, 3, 4
    print(X)            # l'istruzione print
```

```
1
2
3
4
```

Nel corpo del ciclo posso usare le istruzioni

```
break    # per uscire immediatamente dal ciclo
         # e proseguire con le istruzioni DOPO
```

```
         # tutto il blocco FOR
continue # per saltare al prossimo elemento della
         # sequenza senza completare le altre istruzioni del blocco indentato
```

[3]:
```
# ESEMPIO con break e continue (a seconda del valore)
for X in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
    if X == 13:
        break      # esco se incontro il valore 13
    if X % 2 == 0:
        continue   # salto tutti i numeri pari
    print(X)
else:
    print('li ho stampati tutti') # solo se ho completato il FOR senza break
```

```
1
3
5
7
9
li ho stampati tutti
```

10.2 Come contare: con il generatore di sequenze range

E' un generatore di sequenze di interi, cioè un oggetto che fornisce un numero per volta | chiamandolo così | | produce uno dopo l'altro la sequenza di valori| |:———————|:–|:—————————————| |`range(fine)` | |**0, 1, 2, 3, 4.... (fine-1)**| |`range(inizio, fine)` | |**inizio, inizio+1, ... , fine-1**| |`range(inizio, fine, incremento)` | |**inizio, inizio+incremento, ..., fine-1**|

In Python 3 ci sono moltissimi generatori, che riducono l'uso della memoria

NOTATE CHE: il valore di `fine` non viene mai prodotto, e che range si comporta proprio come una slice

[4]:
```
## range crea un oggetto che fornisce un nuovo valore ogni volta
## che gliene viene richiesto uno nuovo dal 'for' o da 'list'
R = range(-12,-30,-3)
print(R, type(R) )      # se lo stampo mi mostra che è un oggetto di tipo␣
 ↪'range'
for i in R:             # se lo uso nel FOR mi genera i numeri 1...9
    print(i, end=' ')
list(R)                 # e posso anche usarlo per costruire una lista di valori
```

```
range(-12, -30, -3) <class 'range'>
-12 -15 -18 -21 -24 -27
```

[4]: `[-12, -15, -18, -21, -24, -27]`

10.3 Caso 2: quando NON SAPETE QUANTE iterazioni/ripetizioni fare ALLORA è meglio usare WHILE

```
prima dobbiamo inizializzare le variabili usate nella condizione
while condizione_vera:
    # blocco eseguito SOLO SE LA CONDIZIONE E' VERA
    blocco di codice da ripetere
    # FONDAMENTALE!!!! DA NON DIMENTICARE!!!
    aggiornate SEMPRE i dati della condizione
    altrimenti il ciclo NON SI FERMA PIù
    perchè la condizione resta sempre vera
else:       # parte opzionale (come per il for)
    blocco di codice eseguito solo se si termina il ciclo
    perchè la condizione è diventata False
    ovvero NON siamo usciti con break
codice a seguire
```

[69]:
```
# ESEMPIO
X = 0                   # all'inizio X vale 0
while X < 20:           # ripeto ma solo se X è minore di 20
    X += 1              # aggiorno la condizione incrementando X (conviene farlo␣
 ↪subito)
    if X == 12:         # quando arrivo a 12
        break           # esco
    if X % 2 == 0:      # altrimenti se X è pari
        continue        # lo salto e passo al prossimo
    print(X, end=' ')   # stampo X (seguito da spazio invece che da accapo)
else:                   # se esco perchè X non è più < 20
    print('\nli ho stampati tutti')       # lo dico
print('\ninizio del codice seguente')   # e poi continuo
```

```
1 3 5 7 9 11
inizio del codice seguente
```

FDPLEZ3

10.4 Quante stampe fa il programma ?

[194]:
```
# Soluzione
# Quante volte il programma stampa qualcosa?
for X in range(50, -1, -7):                    # 50, 43, 36, 29, 22, 15, 8, 1
    if X % 5 == 0:                             # salto 50, 15
        continue
    if X // 7 < 5:                             # quando X < 35=7*5 esco
        break
    print(X)                                   # stampo solo 43 e 36
else:
    print("Ho finito")                         # non lo stampo perchè sono␣
 ↪uscito con break
print("e ora basta")                           # questo lo stampo, il for è␣
 ↪finito
```

```
43
```

```
36
e ora basta
```

[195]:
```python
# Soluzione
# Quante volte il programma stampa qualcosa?
N = 0
while 2**N < 5000:                                  # fino a N==12 compreso
    if N % 3 == 0:                                  # salto i multipli di 3 (0, 3,␣
 ↪6, 9, 12)
        N += 2                                      # e il valore seguente  (1, 4,␣
 ↪7, 10)
        continue
    if N > 12:                                      # questo non succede mai
        break
    print(2**N, N)                                  # stampo potenza e indice
    N += 1                                          # aggiorno N
else:
    print("Ho finito")                              # stampo, NON sono uscito col␣
 ↪break
print("facciamo un intervallo?")                    # il while è finito, si stampa
```

```
4 2
32 5
256 8
2048 11
Ho finito
facciamo un intervallo?
```

11 I contenitori: liste, tuple, dizionari, insiemi

	tipo	parentesi	esempio	che fa	indicizzata?	modificabile?
lista	`list`	`[]`quadre	`[1, 2, 3, 4]`	lista di elementi eterogenei	indicizzata	modificabile
tupla	`tuple`	`()`tonde	`(1, 2, 3, 4)`	n-upla di elementi eterogeneri	indicizzata	NONmodificabile
insieme	`set`	`{}`graffe	`{1, 2, 3}`	insieme di elementi eterogenei im-mutabilisenza ripetizioni	NON indicizzatoe in ordine non fisso	modificabile
dizionario	`dict`	`{}`graffe	`{'a': 1, 'b': 2}`	coppiechiave unica : valore	indicizzato sulle chiavi	modificabile

[70]:
```
# Esempio: costruisco una lista
lista_valori = [2, 5, 7, 23, 45, 2, 7, 23, 'tre' ]
lista_valori
```

[70]:
```
[2, 5, 7, 23, 45, 2, 7, 23, 'tre']
```

[71]:
```
# costruisco una tupla con gli elementi della lista
tupla_valori = tuple(lista_valori)
tupla_valori
```

[71]:
```
(2, 5, 7, 23, 45, 2, 7, 23, 'tre')
```

[72]:
```
# costruisco un insieme con gli elementi della lista
set_valori   = set(lista_valori)
set_valori   # solo elementi unici in ordine non fisso
```

[72]:
```
{2, 23, 45, 5, 7, 'tre'}
```

[85]:
```
# costruisco un dizionario    chiave (unica) -> valore
dizionario   = { 'a': 1, 'b': 2, 'c': 3, 45: 17, (2,5):'paperino', 'c':␣
 ↪'Minnie'}
dizionario   # le chiavi sono uniche ed appare l'ultima se ci sono doppioni
```

[85]:
```
{'a': 1, 'b': 2, 'c': 'Minnie', 45: 17, (2, 5): 'paperino'}
```

[86]:
```
## le liste sono indicizzate E **modificabili**
print(lista_valori[4])   # ne stampo uno (in posizione 4)
lista_valori[5] = 666    # cambio un elemento in posizione 5
print(lista_valori)      # la lista è cambiata
```

```
45
[2, 5, 7, 23, 45, 666, 7, 23, 'tre']
```

[88]:
```
## gli insiemi NON sono indicizzati e sono **modificabili**
print(set_valori)
print(set_valori.pop())   # estraggo un elemento (a caso) con 'pop'
```

```
{7, 45, 'tre', 23, 666}
7
```

[89]:
```
set_valori.add(666)       # aggiungo un elemento
print(set_valori)
```

```
{45, 'tre', 23, 666}
```

[90]:
```
# leggo l'elemento in posizione 4
set_valori[4]             # ERRORE!!! gli insiemi NON sono indicizzati!
```

```
---------------------------------------------------------------------------
```

```
TypeError                                 Traceback (most recent call last)
Cell In[90], line 2
      1 # leggo l'elemento in posizione 4
----> 2 set_valori[4]             # ERRORE!!! gli insiemi NON sono indicizzati!

TypeError: 'set' object is not subscriptable
```

[91]:
```
## le tuple sono indicizzate MA **immutabili**
print(tupla_valori[4])  # ne leggo un valore
tupla_valori[5] = 666   ### ma se provo a modificarla ERRORE!
```

```
45
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[91], line 3
      1 ## le tuple sono indicizzate MA **immutabili**
      2 print(tupla_valori[4])  # ne leggo un valore
----> 3 tupla_valori[5] = 666   ### ma se provo a modificarla ERRORE!

TypeError: 'tuple' object does not support item assignment
```

[92]:
```
# i dizionari sono indicizzati dalle chiavi e **modificabili**
print(dizionario)
print(dizionario.keys())          # estraggo le chiavi -> generatore
print(list(dizionario.keys()))    # estraggo le chiavi -> lista
```

```
{'a': 1, 'b': 2, 'c': 'Minnie', 45: 17, (2, 5): 'paperino'}
dict_keys(['a', 'b', 'c', 45, (2, 5)])
['a', 'b', 'c', 45, (2, 5)]
```

[93]:
```
print(dizionario['a'])      # stampo il valore associato alla chiave 'a'
dizionario['paperino']  = 42  # aggiungo una coppia chiave -> valore
dizionario[(2, 5)]      = 3.14  # modifico una coppia chiave -> valore
del dizionario['c']         # elimino la coppia con chiave 'c'
print(dizionario)           # il dizionario è cambiato
```

```
1
{'a': 1, 'b': 2, 45: 17, (2, 5): 3.14, 'paperino': 42}
```

[94]:
```
dizionario['pluto']         # se la chiave non c'è ERRORE!!!
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[94], line 1
----> 1 dizionario['pluto']         # se la chiave non c'è ERRORE!!!
```

```
KeyError: 'pluto'
```

[95]:
```
# posso controllare se la chiave è presente con 'in'
print('pluto' in dizionario)
print(list(dizionario.values()))   # e posso estrarre i soli valori
```

```
False
[1, 2, 17, 3.14, 42]
```

[96]:
```
# Che succede se metto due volte la stessa chiave 'uno'?
print({ 'uno':1, 'due':2, 'uno':11 })
# appare solo l'ultimo valore perchè le chiavi sono UNICHE
print(dict([('a',3),(17,'paperino'), ('a',33)]))  # conversione da lista di␣
 ↪coppie a dict
```

```
{'uno': 11, 'due': 2}
{'a': 33, 17: 'paperino'}
```

[97]:
```
# come stampare gli elementi dei diversi contenitori?
print(lista_valori)
for elemento in lista_valori:
    print(elemento, end=' ')   # stampa seguita da spazio invece che '\n'
print('\t\tlista_valori')
```

```
[2, 5, 7, 23, 45, 666, 7, 23, 'tre']
2 5 7 23 45 666 7 23 tre                lista_valori
```

[98]:
```
print(set_valori)
for elemento in set_valori:
    print(elemento, end=' ')
print('\t\tset_valori')
```

```
{45, 'tre', 23, 666}
45 tre 23 666           set_valori
```

[99]:
```
print(tupla_valori)
for elemento in tupla_valori:
    print(elemento, end=' ')
print('\t\ttupla_valori')
```

```
(2, 5, 7, 23, 45, 2, 7, 23, 'tre')
2 5 7 23 45 2 7 23 tre          tupla_valori
```

[100]:
```
# Python permette di eseguire più assegnamenti assieme
# e il metodo items dei dizionari produce ciascuna delle coppie (chiave,valore)
print(list(dizionario.items()))
for chiave,elemento in dizionario.items():
    print(chiave,elemento)
print(dizionario.items())       # produce un oggetto 'generatore' di coppie
```

```python
print(list(dizionario.items())) # con list creo una lista che li contiene
```

```
[('a', 1), ('b', 2), (45, 17), ((2, 5), 3.14), ('paperino', 42)]
a 1
b 2
45 17
(2, 5) 3.14
paperino 42
dict_items([('a', 1), ('b', 2), (45, 17), ((2, 5), 3.14), ('paperino', 42)])
[('a', 1), ('b', 2), (45, 17), ((2, 5), 3.14), ('paperino', 42)]
```