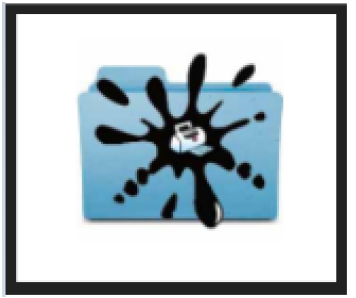
PYTHON

Mini Guida (Prima Parte)

Introduzione al linguaggio

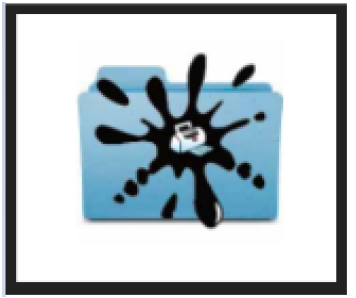


- Python è un linguaggio per la programmazione e lo scripting implementano sotto licenza Open Source; una delle sue caratteristiche più rilevanti è quella di essere multi-purpose e multi-paradigma, questo significa che potrà essere utilizzato per scopi tra loro differenti e sfruttando diversi approcci per lo sviluppo delle applicazioni basate su di esso.

- Per quanto riguarda gli ambiti di utilizzo di Python, lo si potrà impiegare sia per la creazione di software destinati all'installazione su terminale che per la realizzazione di soluzioni per il Web.

- Dal punto di vista dei paradigmi di riferimento, esso si basa innanzitutto sulla programmazione orientata agli oggetti (paradigma Object Oriented), ma prevede anche costrutti derivanti dall'approccio funzionale e da quello imperativo.

Introduzione al linguaggio

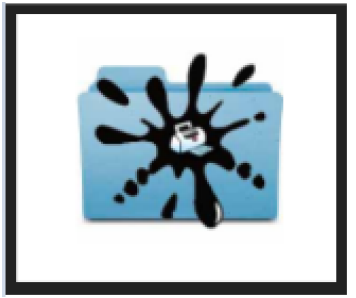


- Python è inoltre un linguaggio multi-piattaforma, è stato infatti concepito per poter essere impiegato su tutti i sistemi operativi più diffusi, Windows, distribuzioni basate sul Kernel Linux e (Mac) OS X.

- La sintassi di questo linguaggio, pensata per essere il più possibile essenziale, ricorda in parte quella di una soluzione come Perl, ma per la sua concezione sono stati utilizzati anche costrutti mediati da C, C++ e Java.

- Da parte sua, invece, Python ha influenzato in vario modo la progettazione di altri linguaggi ampiamente utilizzati, e spesso destinati a contesti differenti, come Java, Ruby e addirittura Swift, strumento creato di recente da Apple per la creazione di applicazioni destinate a iOS e il già citato (Mac) OS X.

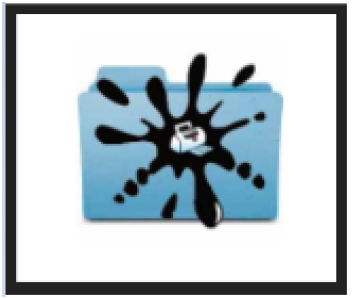
Editor



- Di default Python offre un potente strumento per la programmazione denominato IDLE, esso mette a disposizione un'interfaccia grafica con la quale lanciare istruzioni basate sul linguaggio e creare file eseguibili con estensione ".py".

- Vengono inoltre fornite feature avanzate per incrementare il livello di produttività delle sessioni di sviluppo come per esempio i suggerimenti per il completamento delle istruzioni, l'evidenzione del codice (syntax highlight) tramite una colorazione differente dei vari costrutti, un debugger per i test sugli script, funzionalità per la ricerca all'interno di sorgenti (anche con supporto per le espressioni regolari) e tool per l'indentazione del listato.

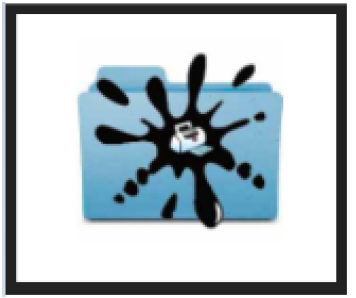
Editor



- Una volta avviato IDLE si potrà lanciare immediatamente un primo comando basato su Python per cominciare ad analizzarne la sintassi di base; le istruzioni dovranno essere scritte dopo i simboli ">>>" che indicano la modalità interattiva, cioè la possibilità di comunicare con l'interprete (si tratta di un'interfaccia del tutto simile alla riga di comando del prompt di Windows)

- Simulatore http://www.pythontutor.com/visualize.html#mode=edit

I commenti



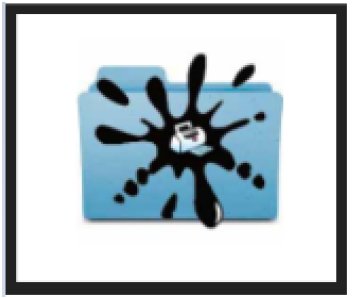
- Come qualsiasi altro linguaggio di programmazione che si rispetti anche Python supporta i commenti, cioè delle annotazioni che lo sviluppatore potrà inserire all'interno del sorgente per incrementarne il livello di leggibilità.

- Grazie a dei caratteri appositamente concepiti per la loro delimitazione, l'interprete del linguaggio riconoscerà i commenti come tali evitando che essi entrino nel flusso di esecuzione del codice e siano visibili all'interno degli output.

- E' possibile distinguere tra commenti a linea singola e commenti multilinea; i commenti a linea singola vengono ospitati su una sola riga e introdotti tramite un delimitatore iniziale che è il carattere cancelletto (#):

```
#Questo è un commento su singola linea.
# Anche questo è un commento su singola linea.
# Basta con i commenti su singola linea!
```

I commenti



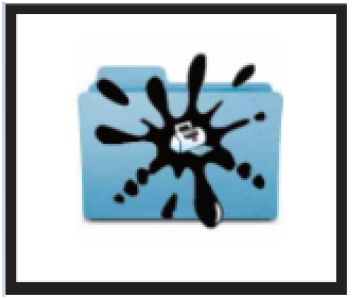
- I commenti multilinea, cioè quelli che si estendono su più righe, vengono invece delimitati tramite tre apici singoli posti sia in apertura che in chiusura del commento:

```
'''
Questo è un commento
multilinea.
'''
```

- Per le ragioni esposte è consigliabile, pertanto, l'utilizzo dei commenti linea singola anche per il multilinea, come nell'esempio seguente:

```
# Questo
#è un commento
#multilinea.
```

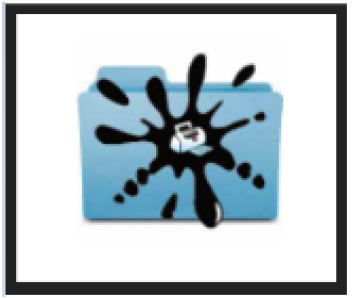
"Ciao Mondo!"



- Come prima interazione con il linguaggio, una volta lanciato IDLE, digitiamo la seguente istruzione e osserviamo l'output generato:

```
# Il mio primo comando in Python
print ("Ciao Mondo!")
```

"Ciao Mondo!"



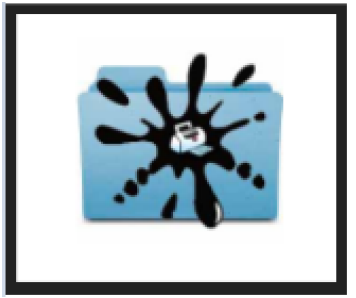
- I costrutti basati sul comando print consentono di stampare una stringa passata come parametro (nel nostro caso "Ciao Mondo!"); come è facile notare l'interprete riconosce la stringa perché delimitata da doppi apici, l'argomento passato a print viene a sua volta delimitato grazie a delle parentesi tonde.

- Il commento iniziale verrà invece ignorato e non parteciperà alla generazione dell'output.

- L'uso delle parentesi è fondamentale quando si programma con il ramo 3.x del linguaggio, in precedenza infatti, come accade con il ramo 2.x, era possibile ottenere il medesimo risultato utilizzando un formato come il seguente:

```
print "Ciao Mondo!"
```

Le variabili



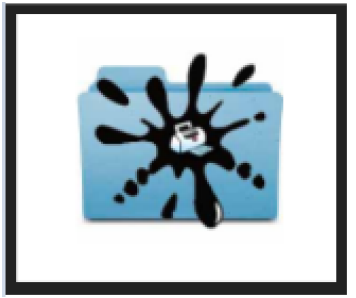
- Le variabili sono dei costrutti comuni a quasi tutti i linguaggi di programmazione, esse nascono dall'esigenza di avere a disposizione uno spazio di memoria al quale associare dinamicamente delle informazioni;.

- Questa operazione per la memorizzazione diretta dei dati consiste in pratica nell'assegnazione di un valore alle variabili.

- Per comprendere il funzionamento di tale processo di assegnazione è possibile digitare le seguenti istruzioni:

```
# Associare un valore stringa ad una variabile
a = "Questa è una variabile"
print(a)
```

Le variabili

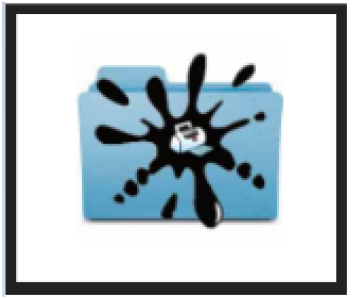


```
# Associare un valore numerico ad una variabile
a = 1000
print(a)
```

- In questo caso invece, l'output prodotto dall'applicazione sarà "1000". Quindi, volendo tirare le somme sulla base di quanto visto fino ad ora, possiamo affermare che una variabile deve essere rappresentata da una stringa, che nel nostro esempio è il solo carattere "a".

- Per assegnare ad essa un valore di dovrà utilizzare il simbolo di uguaglianza ("=") che, nello specifico, agisce come operatore per l'assegnazione.

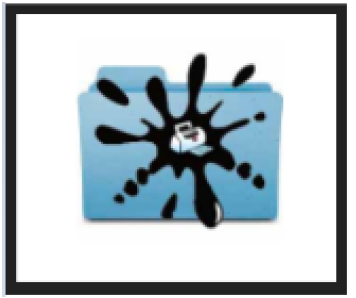
Le variabili



- Esistono alcune regole da rispettare quando si vuole associare un nome ad una variabile che, naturalmente non deve chiamarsi forzatamente "a", esse in sostanza escludono determinate casistiche, motivo per il quale il nome di una variabile:

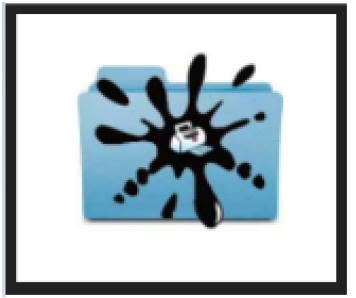
  - non potrà essere un numero, ad esempio non si potrà utilizzare "2" o "22" come nome di variabile;
  - non potrà iniziare con un numero, per cui "2a" o "2aa" non saranno dei formati consentiti dall'interprete di Python;
  - non potrà contenere spazi, quindi non si potrà adottare una variabile chiamata, per esempio, "a a" o "a b";
  - non potrà contenere segni di interpunzione, escludendo la possibilita di formati come "a.a" o "a!a";
  - non potrà contenere simboli, quindi nomi come "a€a" o "a§a" porteranno alla generazione di un errore.

Le variabili



- Sarà invece possibile:
  - utilizzare nomi composti da sequenze di più caratteri, ad esempio "abc", tra cui anche termini di senso compiuto ("ciao");
  - terminare il nome di una variabile con un numero, ad esempio "a2" o "a22";
  - utilizzare una stringa contenente al suo interno delle cifre, come per esempio "a2a";
  - adottare nomi composti da maiuscole e minuscole, per cui una variabile potrà essere chiamata anche "A", "AbC" o "ABC".

Le variabili



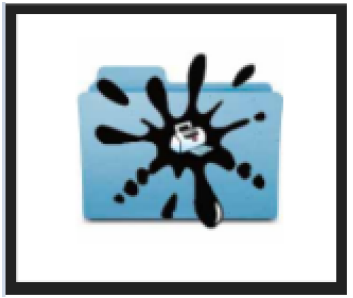
- Per quanto riguarda le regole dei valori utilizzati per l'assegnazione, come risulta chiaro dall'esempio procedente un valore stringa dovrà essere delimitato da doppi apici (a = "Questa è una variabile") mentre un valore numerico no (a = 2).

- Un valore numerico delimitato da doppi apici (come per esempio a = "10") verrà interpretato come una stringa.

- Poter comunicare all'interprete la corretta tipologia di valore (o tipo di dato) è fondamentale per il corretto funzionamento di uno script.

- A tal proposito si analizzi l'esempio seguente:

Le variabili

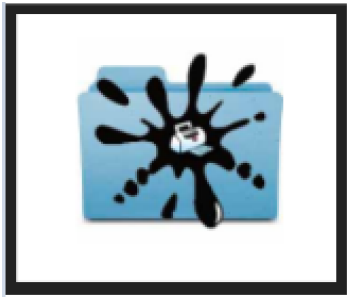


```
# Utilizzare i valori delle variabili
# nei calcoli matematici
x = 0
y = 1000
print(x + y)
```

- La piccola applicazione proposta non fa altro che eseguire un'addizione tra un non valore aritmetico (la cifra "0") e il numerico intero "1000", il risultato dell'operazione sarà naturalmente pari a quest'ultimo valore.

- Ora però riproponiamo lo stesso script nel modo seguente:

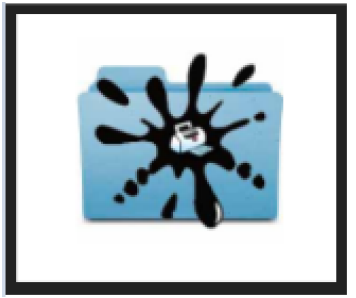
Le variabili



```
# come stringhe
x = "0"
y = "1000"
print(x + y)
```

- In questo secondo caso il risultato dell'esecuzione del codice sarà la stringa "01000", questo perché la presenza dei doppi apici forzerà l'interprete a riconoscere i valori assegnati alle variabili come delle stringhe e il "+" non agirà più come operatore matematico di addizione, ma come operatore per la concatenazione di queste ultime.

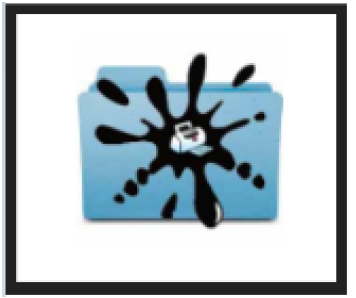
Le variabili



- Lo sviluppatore avrà comunque la possibilità di modificare dinamicamente il tipo di dato associato ad una variabile attraverso i cosiddetti operatori di casting che sono:

  - int(): converte una variabile al tipo di dato intero;
  - str(): converte una variabile al tipo di dato stringa;
  - float(): converte una variabile al tipo di dato float, un numero decimale.

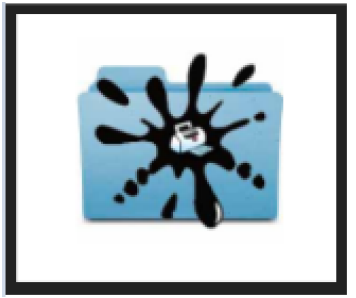
Le variabili



- Anche in questo caso sarà possibile fare riferimento a un esempio:

```
# casting di una variabile stringa
# in una variabile numerica intera
x = "5"
y = 1000
print(int(x) + y)
```

Gli operatori

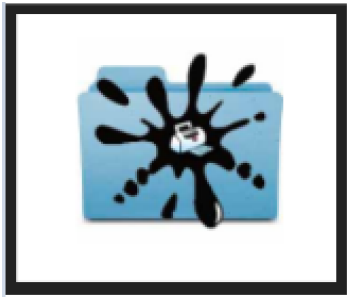


- Gli operatori non sono un'esclusiva di Python, in generale tutti i linguaggi di programmazione e sviluppo dispongono di questi costrutti sintattici;

- E’ possibile definirli come dei simboli attraverso i quali specificare quale regola dovrà essere applicata a degli operandi per l'ottenimento di uno determinato risultato.

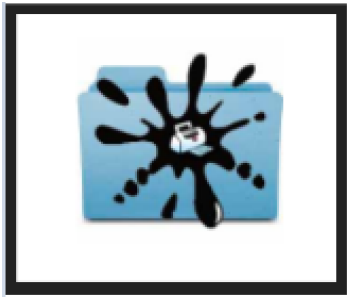
- Gli operatori possono, per esempio, determinare un'assegnazione, ne sono stati descritti alcuni di questo tipo nei capitoli precedenti, consentono di effettuare operazioni matematiche a carico degli operandi (e anche in questo caso sono stati già presentati degli esempi), permettono di effettuare dei confronti tra valori e consentono di incrementare o effettuare decrementi a carico di questi ultimi; tutto dipende dal simbolo utilizzato, dalla tipologia degli operandi impiegati e da alcune regole sintattiche.

Gli operatori



- Gli operatori di confronto producono un risultato sulla base di una comparazione tra operandi.

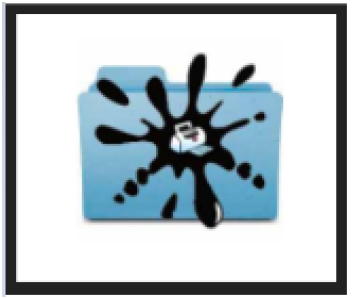
Gli operatori



- I principali operatori di confronto di Python sono:

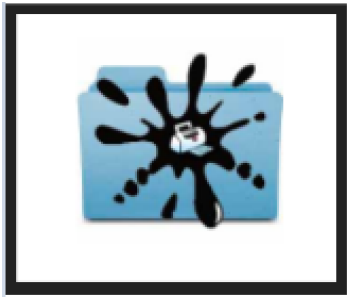
Op.	Descrizione	Esempio
==	Se il valore dei due operandi è il medesimo il confronto restituisce TRUE, altrimenti FALSE.	(3 == 2) restituisce FALSE
!=	Se il valore dei due operandi non è uguale il confronto restituisce TRUE, altrimenti FALSE.	(3 != 2) restituisce TRUE
<>	Se il valore dei due operandi non è uguale il confronto restituisce TRUE, altrimenti FALSE. Come per l'operatore precedente	(3 <> 2) restituisce TRUE
>	Se il valore alla sinistra del simbolo è maggiore di quello alla sua destra restituisce TRUE, altrimenti FALSE.	(2 > 10) restituisce FALSE
<	Se il valore alla sinistra del simbolo è inferiore a quello alla sua destra restituisce TRUE, altrimenti FALSE	(2 < 10) restituisce TRUE
>=	Se il valore alla sinistra dei simboli è maggiore o uguale a quello alla sua destra restituisce TRUE, altrimenti FALSE.	(3 >= 5) restituisce FALSE
<=	Se il valore alla sinistra dei simboli è minore o uguale a quello alla sua destra restituisce TRUE, altrimenti FALSE.	(3 <= 5) restituisce TRUE

Gli operatori



- Gli operatori aritmetici consentono di effettuare calcoli matematici tramite gli operandi.

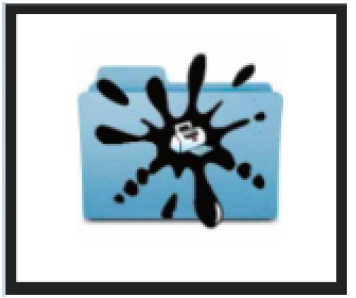
Gli operatori



- Per quanto riguarda questa tipologia in Python abbiamo a disposizione i seguenti operatori:

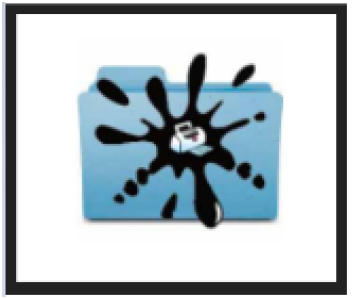
Op.	Descrizione	Esempio
+	L'operatore di **addizione** somma i valori degli operandi.	3 + 2 = 5
-	L'operatore di **sottrazione** decrementa l'operando alla sinistra del simbolo di un valore pari a quello dell'operando alla destra di quest'ultimo.	3 – 2 = 1
*	L'operatore di **moltiplicazione** moltiplica l'operando alla sinistra del simbolo un numero di volte pari al valore dell'operando posto alla sua destra.	3 * 2 = 6
/	L'operatore di **divisione** divide l'operando alla sinistra del simbolo sulla base del valore associato all'operando posto alla sua destra.	4 / 2 = 2
%	L'operatore **modulo** restituisce il resto di una divisione.	5 % 2 = 1
**	L'operatore **esponente** eleva a potenza l'operando alla sinistra del simbolo un numero di volte pari al valore dell'operando posto alla sua destra.	8**2 = 64
//	L'operatore di **arrotondamento** restituisce il risultato di una divisione arrotondandolo al valore intero più prossimo a quello reale.	8 // 3 = 2 oppure 9.5 // 2 = 4.0

Gli operatori



- Gli operatori di assegnazione permettono non solo di attribuire dei valori, ma anche di specificare le modalità di tale attribuzione.

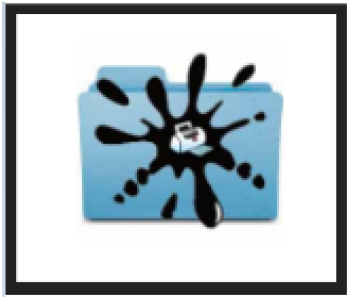
Gli operatori



- Dato che essi trovano ampio utilizzo quando si opera con le variabili, nella tabella seguente ne verranno utilizzate due come riferimento, "x" ed "y", inizialmente assegnate ai valori "1" e "2":

Op.	Descrizione	Esempio
=	Assegna il valore dell'operando alla sinistra del simbolo a quello alla sua destra	z = x + y per cui il valore di z sarà pari a 3
+=	Aggiunge l'operando di destra a quello di sinistra e assegna il risultato a quest'ultimo.	z += x equivale a z = z + x
-=	Sottrae l'operando di destra a quello di sinistra e assegna il risultato a quest'ultimo.	z -= x equivale a z = z - x
*=	Moltiplica l'operando di destra per quello di sinistra e assegna il risultato a quest'ultimo.	z *= x equivale a z = z * x
/=	divide l'operando di sinistra per il valore di quello di destra e assegna il risultato al primo operando.	z /= x equivale a z = z / x
%=	Calcola il resto dei due operatori e assegna il risultato all'operando di sinistra.	z %= x equivale a z = z % x
**=	Calcola un'elevazione a potenza e assegna il risultato all'operatorendo di sinistra.	z **= x equivale a z = z ** x
//=	Restituisce l'arrotondamento di una divisione tra gli operatori e assegna il risultato all'operatorendo di sinistra.	z //= x equivale a z = z // x

Gli operatori

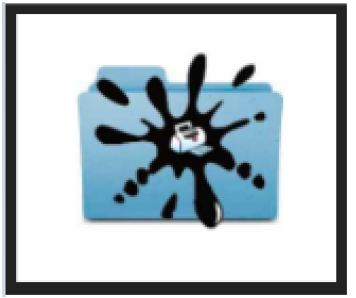


- Gli operatori logici sono dei costrutti sintattici utili al fine di legare due o più condizioni tra di loro, dove per **condizione** si intende una qualunque affermazione che può essere vera o falsa.

- Si ipotizzi per esempio di avere una variabile "x" assegnata al valore TRUE e un'altra, "y", assegnata al valore FALSE:

Op.	Descrizione	Esempio
and	Restituisce TRUE se entrambi gli operatori sono TRUE.	x and y restituisce FALSE
or	Restituisce TRUE se almeno uno dei due operatori è TRUE.	x or y restituisce TRUE
not	Restituisce TRUE se l'operando è FALSE.	not y restituisce TRUE

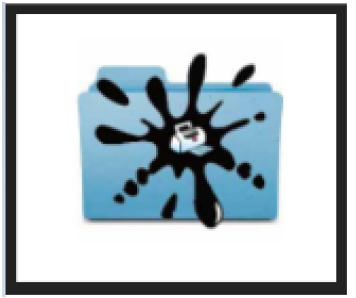
Gli operatori



- Gli operatori di membership potranno essere utilizzati per verificare se un determinato elemento fa parte o meno di un insieme, ad esempio una sequenza alfanumerica:

Op.	Descrizione	Esempio
in	Restituisce TRUE se un valore è presente all'interno di un insime.	Se x = "Ciao mondo" allora print("C" in x) restituisce TRUE
not in	Restituisce TRUE se un valore non è presente all'interno di un insime.	Se x = "Ciao mondo" allora print("Z" not in x) restituisce TRUE

Gli operatori

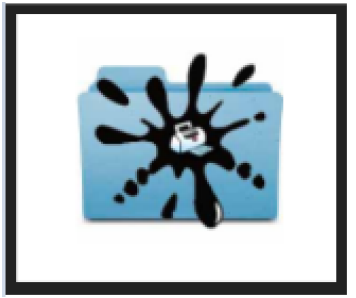


- Gli operatori di identità vengono utilizzati per verificare se due valori sono stati archiviati nella stessa porzione di memoria.

- Si tenga conto del fatto che per Python due valori "uguali" non sono necessariamente anche "identici":

Op.	Descrizione	Esempio
is	Restituisce TRUE se gli operandi fanno riferimento al medesimo oggetto.	Se x = "ciao" e y = "ciao" allora print(x is y) restituisce TRUE perché uguali per valore e identici per tipo di dato
is not	Restituisce TRUE se gli operandi non fanno riferimento al medesimo oggetto	Se x = 3 e y = 3 allora print(x is not y) restituisce FALSE

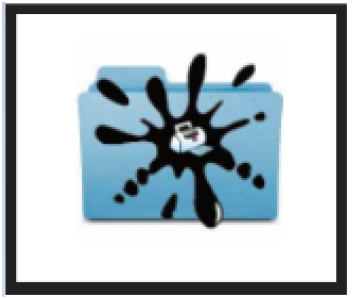
I costrutti condizionali



- Come la maggior parte dei linguaggi per la programmazione e lo sviluppo anche Python mette a disposizione dei coders i cosiddetti costrutti condizionali; in sostanza parliamo di espressioni il cui esito, cioè il risultato prodotto, dipende dalla soddisfazione o meno di una condizione precedentemente definita. Per la definizione di un costrutto condizionale si utilizzano le keywords if, l'unica obbligatoria, elif ed else, gli esempi proposti di seguito ne evidenzieranno il funzionamento.

- La keyword if ha il compito di introdurre una condizione, nel codice seguente quest'ultima si basa sull'uguaglianza di un valore passato interattivamente come parametro a quello assegnato ad una variabile.

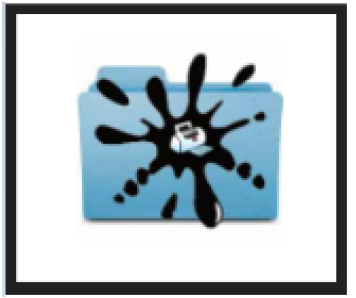
I costrutti condizionali



```
# Utilizzo di if nei costrutti condizionali
# l'applicazione restituirà in output una conferma
# soltanto se il parametro inviato dall'utente
# è identico al valore della variabile x

x = 8
y = int(input("Inserisci un numero intero compreso tra 0 e 10: "))
if y == x:
    print("Il numero inserito è esatto")
```

- Tramite lo script proposto l'utente potrà digitare un numero che verrà impiegato come parametro per valorizzare la variabile "y", nel caso in cui tale valore dovesse essere identico a quello di "x", è cioè pari a "8", allora l'applicazione stamperà il messaggio di conferma "Il numero inserito è esatto", altrimenti nulla verrà restituito in output.

l costrutti condizionali



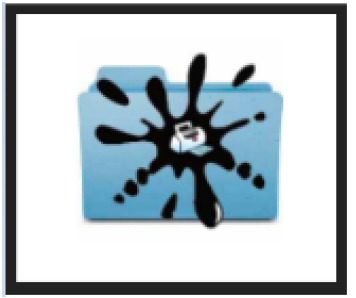
- Il codice proposto, per quanto funzionante, è quindi incompleto, non prevede infatti alcun comportamento per i casi in cui l'uguaglianza richiesta dalla condizione introdotta da if non venga soddisfatta.

- A questo proposito è quindi possibile incrementare le funzionalità previste facendo ricorso alla keyword else.

- else consente di definire un'istruzione alternativa che verrà eseguita quando una condizione imposta tramite if non dovesse verificarsi.

- A titolo di esempio, si implementi lo snippet precedente in questo modo.

I costrutti condizionali

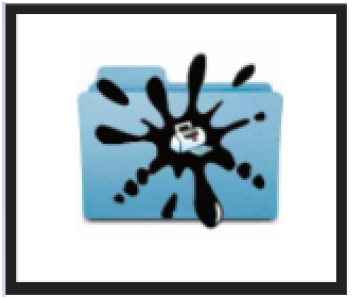


```
# Utilizzo del blocco if/else nei costrutti condizionali
# l'applicazione restituirà in output una conferma
# se il parametro inviato è identico al valore di x
# altrimenti verrà eseguita l'istruzione introdotta da else

x = 8
y = int(input("Inserisci un numero da 0 a 10: "))
if y == x:
    print("Il numero inserito è esatto")
else:
    print("Il numero inserito non è esatto")
```

- L'istruzione introdotta da else dovrà essere preceduta da quest'ultimo seguito dal simbolo dei due punti (":"), esattamente come accade per if.

- Nel caso in cui il valore passato ad "y" sia diverso da "8" allora verrà stampato a video il messaggio "Il numero inserito non è esatto".

l costrutti condizionali

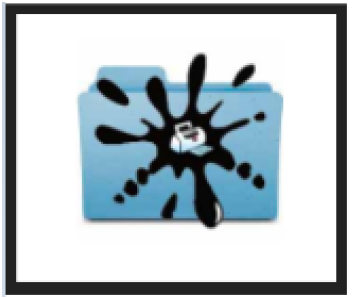


- L'istruzione introdotta da else dovrà essere preceduta da quest'ultimo seguito dal simbolo dei due punti (":"), esattamente come accade per if.

- Nel caso in cui il valore passato ad "y" sia diverso da "8" allora verrà stampato a video il messaggio "Il numero inserito non è esatto".

- Ora l'applicazione proposta risulta più completa rispetto a quella proposta nel primo esempio, cosa accadrebbe però se l'utente dovesse inserire un valore presente al di fuori dell'intervallo consentito, cioè tra "0" e "10"? Per questo caso specifico non è stato ancora definito alcun controllo.

I costrutti condizionali

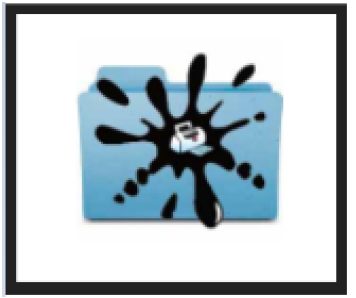


- elif è una parola chiave che, come else, introduce un'istruzione alternativa nei costrutti condizionali, ma l'esecuzione di quest'ultima sarà vincolata ad un'ulteriore condizione.

- Anche in questo caso un esempio pratico consentirà di capire meglio quanto esposto:

```
# Utilizzo del blocco if/elif/else nei costrutti condizionali
# verrà restituita in output una conferma
# se il parametro inviato è identico al valore di x
# altrimenti verrà eseguito un confronto tra i valori di y e z
# o restituita l'istruzione introdotta da else

x = 8
z = 10
y = int(input("Inserisci un numero da 0 a " + str(z) + ":"))
if y == x:
    print("Il numero inserito è esatto")
elif y > z:
    print("Sono consentiti soltanto valori compresi tra 0 e " + str(z))
else:
    print("Il numero inserito non è esatto")
```

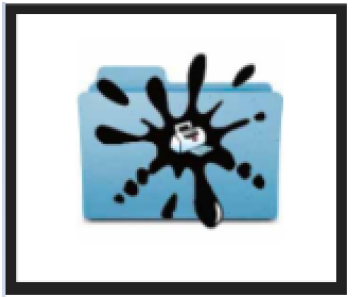
I costrutti condizionali



- In sostanza if introduce una condizione, per soddisfarla il valore di "y" dovrà essere uguale a quello di "x", nel caso in cui ciò non sia vero allora l'applicazione valuterà una seconda condizione, introdotta da elif e anch'essa seguita dal simbolo dei ":", che sarà soddisfatta soltanto se "y" dovesse avere un valore superiore a "z". La mancata soddisfazione di entrambe le condizioni indicate porterà invece all'esito previsto tramite else.

- Dal punto di vista dell'ottimizzazione, dato che quello che si sta realizzando è un programma interattivo, potrebbe essere una buona idea fornire all'utilizzatore degli indizi che gli permettano di scoprire più velocemente l'entità del valore assegnato ad "x" diminuendo il numero di tentativi effettuati.

I costrutti condizionali

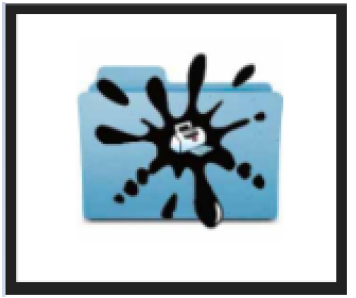


```
In # Utilizzo del blocco if/elif nei costrutti condizionali
# verrà restituita in output una conferma
# se il parametro inviato è identico al valore di x
# altrimenti verrà eseguito un confronto tra i valori di y e x

x = 8
y = int(input("Inserisci un numero da 0 a 10: "))
if y == x:
    print("Il numero inserito è esatto")
elif y > x:
    print("Il numero inserito è superiore a quello atteso")
elif y < x:
    print("Il numero inserito è inferiore a quello atteso")
```

- In pratica, la logica ci suggerisce che l'applicazione potrà funzionare perfettamente prevedendo soltanto tre casi: quello in cui il valore digitato sia esatto, quello in cui esso sia maggiore di quello atteso e quello in cui esso sia invece inferiore.

- Prevedendo tali eventualità in altrettante condizioni non sarà necessario il ricorso ad un'alternativa introdotta con else.

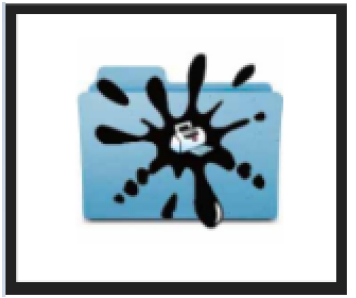
I costrutti condizionali



- In Python i costrutti condizionali prevedono la possibilità degli annidamenti (nested statements), questo significa che si potranno definire delle condizioni all'interno di altre condizioni. A tal proposito si analizzi il seguente esempio:

```
# Annidamento dei blocchi if/elif/else nei costrutti condizionali

x = 8
y = int(input("Inserisci un numero da 0 a 10: "))
if y == x:
    print("Il numero inserito è esatto")
elif y > x:
    print("Il numero inserito è superiore a quello atteso..")
    if (y - x) == 1:
        print("..ma ci sei andato molto vicino.")
    else:
        print("..riprova, sarai più fortunato.")
elif y < x:
    print("Il numero inserito è inferiore a quello atteso")
    if (x - y) == 1:
        print("..ma ci sei andato molto vicino.")
    else:
        print("..riprova, sarai più fortunato.")
```

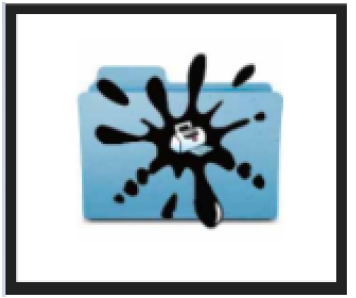
I costrutti condizionali



- La piccola applicazione proposta non si limita a controllare che il valore proposto dall'input dell'utente sia uguale, maggiore o inferiore a quello atteso, ma verifica anche il grado di approssimazione di tale valore a quello esatto.

- Uno scostamento unitario in eccesso o in difetto verrà notificato tramite un'apposita segnalazione aggiuntiva ("..ma ci sei andato molto vicino."), scostamenti più elevati porteranno alla stampa di un'alternativa introdotta tramite else ("..riprova, sarai più fortunato.").

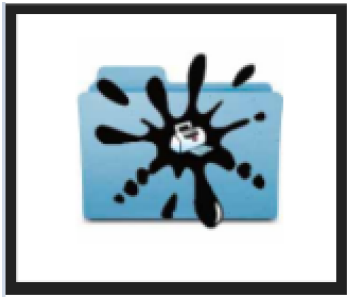
Il ciclo for



- Il ciclo (loop) for è uno dei cicli di iterazione messi a disposizione dal linguaggio Python, si tratta in sostanza di un costrutto, comune anche ad altri linguaggi, che consente di ripetere un'operazione un certo numero di volte, più tecnicamente "iterare una sequenza o un oggetto", fino alla soddisfazione di una determinata condizione (implicita) che terminerà il ciclo.

- Per chiarire tale dinamica sarà possibile proporre un semplice esempio basato sull'impiego delle liste, queste ultime sono un tipo di dato supportato da Python che verrà analizzato nel dettaglio in seguito, per il momento basti sapere che una lista permette di gestire un insieme di valori di diversa natura (stringhe, interi, decimali) delimitati da parentesi quadre e separati tramite una virgola.

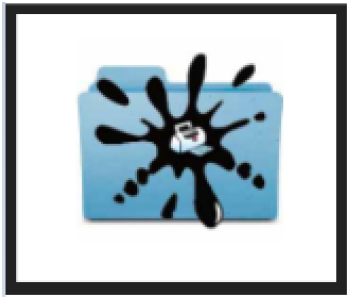
Il ciclo for



- Il codice proposto di seguito permetterà di ciclare e visualizzare in output tutti gli elementi presenti in una lista definita dallo sviluppatore:

```
# Ciclare gli elementi presenti in una lista tramite il ciclo for

# definizione della lista
fibonacci = [1,1,2,3,5,8,13,21,34,55,89,144]

# iterazione dei valori in lista
for val in fibonacci:

    # stampa dei valori iterati
    print(val)
```

Il ciclo for

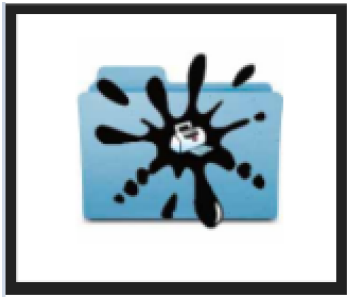


- Nell'esempio mostrato la condizione da soddisfare per la terminazione del ciclo sarà quindi quella di stampare fino all'ultimo valore presente nella lista passata come argomento, nel nostro caso "fibonacci".

- Un fattore sintattico molto importante da tenere a mente riguarda il fatto che nella digitazione di un ciclo for l'engine di Python si aspetta che l'istruzione associata al ciclo venga indentata (si noti infatti la presenza di una tabulazione prima del comando print()), ignorando tale regola in fase di esecuzione si assisterà alla generazione di un errore segnalato tramite la notifica:

SyntaxError: expected an indented block

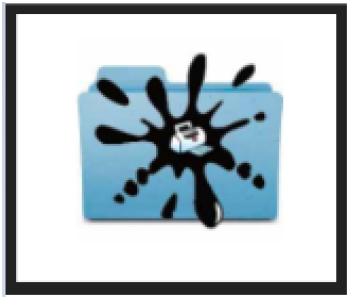
Il ciclo for



- E' interessante notare come il ciclo for possa essere sfruttato anche per eseguire operazioni più complesse rispetto alla semplice stampa a video degli elementi che compongono una sequenza; a tal proposito si potrà modificare la piccola applicazione precedentemente proposta in modo che quest'ultima sommi tutti i valori presenti nella lista restituendo in output il totale ottenuto:

```
# Sommare gli elementi presenti in una lista
# tramite il ciclo for

# definizione della lista
fibonacci = [1,1,2,3,5,8,13,21,34,55,89,144]

# variabile per l'archiviazione della somma
# dei valori in lista
totale = 0

# sommatoria dei valori in lista
for val in fibonacci:
    totale = totale + val

# stampa della somma ottenuta
print(totale)
```

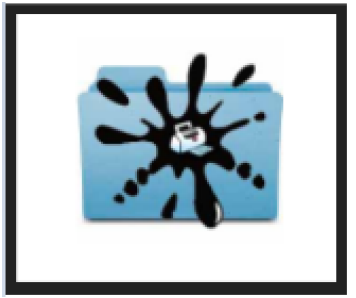
Il ciclo for



- Nel caso specifico la condizione di terminazione del ciclo è il raggiungimento del totale tra tutti i valori degli elementi in lista, ma sarebbe possibile rendere il sorgente più articolato introducendo un controllo basato su if che escluda dalla somma determinati valori:

```
# Sommare gli elementi presenti in una lista
# tramite il ciclo for
# escludendo tutti i valori inferiori a 3

# definizione della lista
fibonacci = [1,1,2,3,5,8,13,21,34,55,89,144]

# variabile per l'archiviazione della somma
# dei valori in lista
totale = 0

# sommatoria dei valori in lista
for val in fibonacci:
   # controllo sul valore degli elementi sommati
   if val > 3:
      totale = totale + val
# controllo sul totale ottenuto
if totale == 369:
   # stampa della somma ottenuta
   print("Il totale della somma è " + str(totale))
else:
   # istruzione alternativa in caso
   # di esito negativo del controllo
   print ("Valore differente da quello atteso.")
```

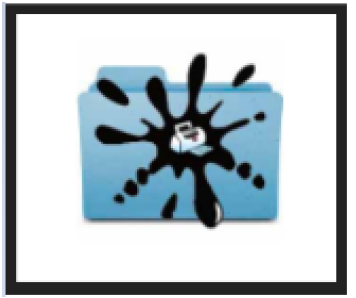
Il ciclo for



- Modificando il codice utilizzato è mutata anche la condizione per la terminazione del ciclo, questa volta infatti dovranno essere sommati tutti gli elementi presenti in lista tranne quelli di valore inferiore a "3".

- L'output dell'applicazione originale era pari a "376", ma dato che nella sequenza sono presenti quattro valori inferiori o uguali a "3" ("1, 1, 2 e 3"), dopo le modifiche apportare il risultato sarà "369", cioè l'output iniziale meno il totale dei valori ora esclusi.

Il ciclo for

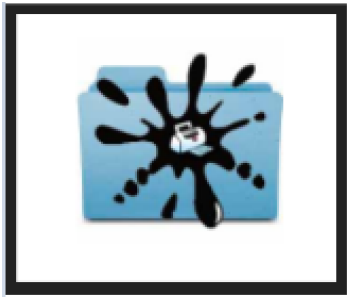


- La funzione range() permette di non dover digitare sequenze di valori eccessivamente lunghe e consente di definire un intervallo di valori sulla base di due parametri: un valore iniziale e un valore di terminazione.

- E’ poi possibile passare alla funzione un terzo argomento (step) che rappresenta le posizioni che dovranno essere ignorate all'interno dell'intervallo. A titolo di esempio si esegua l'istruzione seguente:

```
# Stampa di un intervallo di valori compresi tra 5 e 20

for val in range(5,20):
    print (val)
```

Il ciclo for



- Il risultato ottenuto sarà rappresentato da tutti i valori interni all'intervallo che va da "5" a "19", in quanto valore di terminazione "20" viene considerato invece al di fuori dell'intervallo e non verrà restituito dall'applicazione. Detto questo, è possibile definire anche uno step che influenzerà direttamente la generazione dell'output:

```
# Stampa di un intervallo di valori compresi tra 5 e 20
# escludendo due posizioni ad ogni iterazione

for val in range(5,20,2):
    print (val)
```

- Questa volta la presenza del terzo parametro permetterà di escludere dal risultato un numero di posizioni pari al valore associato a tale argomento per ogni iterazione del ciclo, motivo per il quale non si otterrà più la sequenza "5 6 7 8 .. 19" ma "5 7 9 11 .. 19".