Linguaggio Python

Python

- **Python: creato nel 1991 da Guido Van Rossum**
  - **Il nome viene dal gruppo inglese *Monty Python***
    Van Rossum "I was in a slightly irreverent mood"
- ***Idea di base*: semplicità ed estensibilità**
  - **Sintassi concisa e chiara**
  - **Esalta la leggibilità del codice**
  - **Archivio di moduli vasto e stabile**

http://docs.python.org/

Grande popolarità recente – uso crescente per diverse applicazione

Esecuzione

- **Interprete Python: compilatore + interprete**
- **Modello a Bytecode**
  - **Macchina Virtuale Python (PVM)**

- ***python filename.py***
- **python -c '*statement* '**
- **python (shell interattiva con prompt >>>)**

  >>> print(“Hello World”)
  Hello World
  >>>

Comodi per testare singoli statement o brevi blocchi di codice

Ctrl^D per uscire

Blocchi di codice e statement

Il blocco di codice è caratterizzato SOLTANTO da:
1) simbolo “:” (due punti)
2) indentazione del codice

```
#file hello_world.py
print(“Hello world\n”);
 a = 0; b = 3
 while b < 10:
     print(a, b)
     a, b = 2*b, a+b   #assegnamento valori multipli
 print(b, “End”)
```

Punto e virgola (;)
- per comandi sulla stessa riga
- opzionale per concludere uno statement

```
#Stampa
Hello world
0,3
6,3
6,9
15 End
```

Scheletro di uno script Python

#!/usr/bin/python

<Direttive di importazione moduli esterni>

Uso di clausola import

<Sequenza di statement>

Direttive di importazione moduli

- **Direttiva di importazione moduli**

  Statement import che importa un modulo Python (predefinito o generato da utente)

- **La direttiva di importazione del modulo X può avere diversi formati:**
  - **import X**
  - **from X import a, b, c**
  - **from X import ***

Direttive di importazione moduli

- **import X**

importa il modulo X, a cui crea un riferimento nel namespace corrente: useremo *X.a* per riferirci all'oggetto *a* definito nel modulo X

- **from X import a, b, c**

Importa il modulo X e crea riferimenti nel namespace corrente agli oggetti a, b e c definiti in X: useremo direttamente *a, b* e *c (non X.a)*

- **from X import ***

Importa il modulo X e crea riferimenti nel namespace corrente a tutti gli oggetti pubblici definiti nel modulo X: useremo *a* (non *X.a*)

Tipi di dato

Sono definiti i seguenti tipi di dato builtin

- **Numerics: valori numerici (int, float, complex)**
- **Sequence: sequenze di oggetti (str, list, tuple, range)**
- **Set: insiemi (set, frozenset)**
- **Dict: dizionary di coppie key->value**
- **Iterator: iteratori su "contenitori" di oggetti**
- **File, Classi, etc..**

In Python tutte le variabili sono implementate attraverso il concetto di oggetto

>>> type(30) → <class 'int'>

Tipo di dato

- **Attenzione: in Python, non vengono utilizzati caratteri speciali per distinguere il *tipo* di dato contenuto in una variabile (come a volte succede in Perl – es @ per array)**
- **Il tipo di dato non deve essere dichiarato esplicitamente**
  - **Tipizzazione dinamica**
- **ATTENZIONE: se si utilizza lo stesso nome per una nuova variabile, viene sovrascritto il contenuto della variabile precedente**

Tipi numerics

- **In Python esistono 3 tipi di dato numerici:**
  - **Numeri interi lunghi (long)**
    - **Lughezza arbitraria (no limiti teorici, ma pratici: memoria disponibile)**
  - **Numeri float (virgola mobile)**
  - **Numeri complessi**
- **Limiti stampabili tramite il modulo sys**
  - **sys.float_info, sys.float_info.max**
- **NOTA: i Booleani sono considerati un sotto tipo (sottoclasse) di interi**

Tipi numerics

```
>>> import sys
>>> sys.float_info
>>> print(sys.float_info.max)
>>> sys.int_info
sys.int_info(bits_per_digit=30, sizeof_digit=4)
sys.int_info(bits_per_digit=15, sizeof_digit=2)
```

Memorizzati di default in 2^30 in architetture a 64 bit ma possono crescere oltre

```
>>> print(sys.maxsize)
9223372036854775807
>>> print(sys.maxsize+1)
9223372036854775808
```


Numeri float e complex

- I numeri float - implementati in modo simile al C
  - Separatore parte intera/decimale: “.”
- I numeri complessi hanno una parte reale ed una immaginaria, manipolabili singolarmente
- Interpretazione come numero complesso:
  - [Numero reale +] numero reale con suffisso j: z = 10 + 20j; z = -4j
  - z.real: parte reale → z.real 10.00
  - z.imag: parte immaginaria → z.real 20.00

Operazioni sui tipi numerici

- Operazioni aritmetiche standard: +, -, *, /
- Divisione: / - Divisione intera: //
  - 7.0 / 2.0 → 3.5 ; 7.0 // 2.0 → 3.0 ; 7 / 2 → 3
- Resto divisione tra interi: %
- Valore assoluto: abs()
- Numero complesso coniugato: conjugate()
- Elevamento a potenza: pow(), **
- Arrotondamento: math.trunc(), math.floor(), math.ceil(), round()

Sequence e oggetti non ordinati

Sequence: categoria di tipi di dato

- In Python esistono diversi tipi sequence: implementano il concetto di sequenza ordinata di elementi, con caratteristiche differenti
  - Stringhe
  - Liste
  - Tuple
  - Range

Notare che
manca il tipo Vettore/Array

Gruppi non ordinati di elementi:

- Set
- Dictionary

Stringhe

- Le stringhe sono identificate da caratteri racchiusi tra quotes:
  - single quotes (' ') oppure double quotes (“”)
- In Python non c'è alcuna differenza tra ' e “
  - Ovviamente, è necessario utilizzare lo stesso tipo di apici per l'apertura e la chiusura della stringa
- Se si usa un quote, non è necessario effettuare l'escape dell'altro tipo di quote
  - '”Prova”', “'Prova'”

Caratteri speciali

Per introdurre un carattere speciale in una stringa si utilizza il backslash (\) → sequenza speciale di escape

\n: newline

\r: carriage return

\t: tabulazione

\\: un singolo carattere backslash \

\”: un singolo carattere di doppi apici ”

ES. ' He said:”WOW!” ' oppure “ He said:\”WOW!\” “

Stringhe raw

Una ‘r’ prima delle “” quote identifica una stringa raw, che non viene interpretata quando si stampa

Esempi di stringhe raw

Espressione	Risultato stampato
print(“12\t6\t3”)	12 6 3
print(r“12\t6\t3”)	12\t6\t3
print(r"Prova: \n Ciao")	Prova: \n Ciao
print("Prova: \n Ciao")	Prova: Ciao
print('He said \\”Hi\\”')	He said \”Hi\”
print(r'He said \\”Hi\\”')	He said \\”Hi\\”
print("He said \"Hi\"")	He said "Hi"
print(r"He said \"Hi\"")	He said \"Hi\"

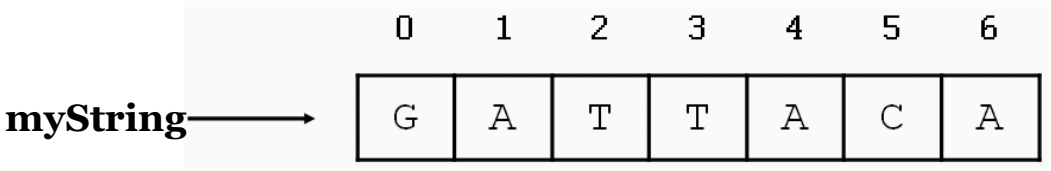
No newline

Newline

Stringhe: sequenze di caratteri

Ogni stringa è memorizzata nella memoria del calcolatore come una lista di caratteri

>>> myString = “GATTACA”



Accesso ai singoli caratteri

E' possibile accedere ai singoli caratteri utilizzando gli indici tra parentesi quadre

```
>>> myString = "GATTACA"
>>> myString[0]
'G'
>>> myString[1]
'A'
>>> myString[-1]
'A'
>>> myString[-2]
'C'
>>> myString[7]

Traceback (most recent call last):
  File "<stdin>", line 1, in ?
IndexError: string index out of range
```

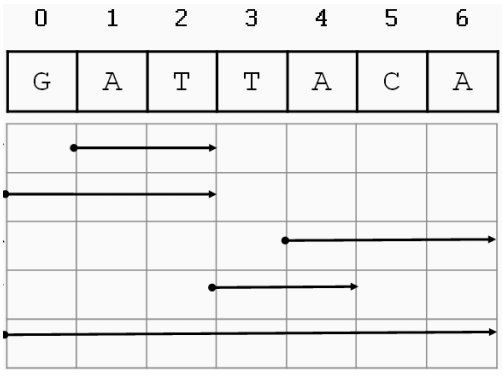
Gli indici negativi iniziano dalla fine della stringa e crescono verso sinistra

Non è come il Perl...

Accesso a sottostringhe (slicing)

E' possibile accedere a sottostringhe (slice) utilizzando sempre l'accesso attraverso indici

```
>>> myString = "GATTACA"
>>> myString[1:3]
'AT'
>>> myString[:3]
'GAT'
>>> myString[4:]
'ACA'
>>> myString[3:5]
'TA'
>>> myString[:]
'GATTACA'
```



Operatore di concatenazione

- Disponibilità di svariati operatori sulle stringhe, tra cui l'operatore di concatenazione (+)

```
>>> firstname = “Dave”
>>> myname = firstname + " " + "Hollinger"
```

- Non è possibile concatenare stringhe con numeri (come invece avviene in Java)
  - Non c'è conversione implicita

```
>>> num = 13
>>> foo = “The number is “ + str(num) + “\n”
```

Ritorna una stringa che rappresenta l'oggetto passato in input

Operatore di ripetizione

Operatore di ripetizione di stringhe (*)

stringa*numero_intero

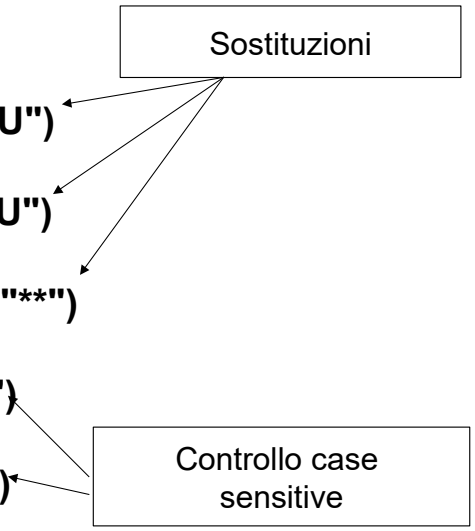
Espressione:	Valore:
“M” * 4	'MMMM'
“Hello” * 2	'HelloHello'
“Joe” * (5 – 2)	'JoeJoeJoe'
'M'*'M'	TypeError: can't multiply sequence by non-int of type 'str'

Metodi definiti sulle stringhe

- len(s): ritorna la lunghezza della stringa s
- s.lower(), s.upper(): ritornano una copia della stringa s con lettere minuscole, maiuscole
- s.count(substr): ritorna il numero di occorrenze della sottostringa substr in s
- s.find(substr): ritorna l'indice della prima occorrenza della sottostringa substr in s
- s.replace(sub1,sub2): rimpiazza le occorrenze della sottostringa sub1 con sub2 in s
- Tantissimi altri – per esempi:
  http://docs.python.org/library/stdtypes.html

Esempi utili

```
>>> len('GATTACA')
7
>>> "GATTACA".lower()
'gattaca'
>>> "GATTACA".replace("G", "U")
'UATTACA‘
>>> "GATTACA".replace("A", "U")
'GUTTUCU'
>>> "GATTACA".replace("AT", "**")
'G**TACA'
>>> "GATTACA".startswith("G")
True
>>> "GATTACA".startswith("g")
False
```



Split e Join

Il metodo s.split(sub) suddivide una stringa s in una sequenza di elementi separati da *sub*

```
>>> '1+2+3+4+5'.split('+')
['1', '2', '3', '4', '5']
>>> 'Using the default'.split()
['Using', 'the', 'default']
```

Ritorna una **lista** di elementi

Il sub di default è lo spazio

Il metodo join() è usato per unire una sequenza (lista o tupla) di stringhe con separatore specificato

```
>>> seq = ['1', '22', '3', '44', '5']
>>> sep = '+'
>>> sep.join(seq)
'1+22+3+44+5'
```

Invocato sul separatore e non sulla sequenza

Le stringhe sono... immutabili!

Attenzione*: *in Python le stringhe non possono essere modificate!
Per la modifica è necessario crearne di nuove

```
>>> s = "GATTACA"
>>> s[3] = "C"
TypeError: 'str' object does not support item assignment

>>> s = s[:3] + "C" + s[4:]
>>> s
'GATCACA'
```

Riassegno la variabile s

E il metodo replace?

Le stringhe sono... immutabili!

In Python i metodi come replace non modificano direttamente la stringa, ma ne restituiscono una nuova **→ per riutilizzarla è necessario assegnarla a un riferimento**

```
>>> s = s.replace("G","U")
>>> s
'UATTACA'
```

```
>>> s.replace("G","U")
'UATTACA
>>> s
'GATTACA'
```

```
>>> sequence = "ACGT"
>>> new_sequence = sequence.replace("A", "G")
>>> new_sequence
'GCGT'
```

→ESERCIZIO
esercizio01.py

Liste

- **Le liste sono contenitori generici di oggetti di qualunque tipo (dynamic typing)**
- **Possono essere sia omogenee che eterogenee**
- **Definizione: elenco di oggetti separati da virgole e delimitato da parentesi quadre**
  - **list = [ ]**
  - **list = [1, 2, 3, 4]**
  - **list = [1, "ciao", 2]**
  - **list = [1, [2, 3], 4]**

Lista vuota

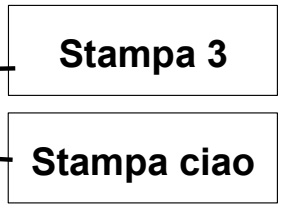
Il secondo elemento è una lista a sua volta!

In altri linguaggi (ad es. Perl), [1, [2, 3], 4] equivale a [1, 2, 3, 4]

Accesso elementi di una lista

- Ciascun elemento di una lista è una variabile contenente un dato del tipo corrispondente
- Per il resto, la sintassi di accesso è simile al C
  - Gli indici di un array partono da 0

```
wt = [1, "ciao", 3, 4]
print(wt[2])
print(wt[1])
```

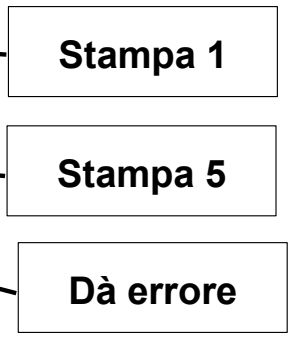


- Si possono usare indici negativi; -1 è l'ultimo elemento, -2 il penultimo, e così via (come per le stringhe)

Liste multidimensionali

- Le liste multidimensionali sono definiti come liste di liste *(non necessariamente di pari lunghezza!)*
- Si usano più indici fra parentesi quadre per l'accesso agli elementi

```
wt = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
print(wt[0][0])
wt = [ 1, [4, 5, 6], 7 ]
print(wt[1][1])
print(wt[0][1])
```



Liste e code

- La lista può essere gestita come una coda
- Metodo append dell'oggetto lista:
  - lista.append(oggetto): appende l'oggetto in fondo alla lista
- Metodo pop dell'oggetto lista:
  - lista.pop(indice): estrae l'oggetto in posizione *indice* dalla lista
  - lista.pop(): estrae l'ultimo elemento della lista
- Metodo insert dell'oggetti lista:
  - lista.insert(indice, oggetto): inserisce l'oggetto nella posizione indicata dall'indice
  - lista.insert(len(lista),oggetto) equivale alla append

Liste e stack

- Gli stessi metodi consentono di gestire facilmente la lista come uno stack
  - Equivalente a una coda a gestione LIFO
- Metodo append:
  - lista.append(oggetto): appende l'oggetto in fondo alla lista
- Metodo pop:
  - lista.pop(): estrae l'oggetto in coda alla lista

Slicing delle liste

- Come per le stringhe, il Python permette la gestione diretta di porzioni di lista (slicing)
- L'operatore di range : viene usato per specificare un intervallo contiguo di indici di una lista

```
wt = [1, 2, 3, 4, 5]
# notazione [start:stop]
wt_slice = wt[1:3]    → [2,3]
# notazione [start:stop:increment]
wt_slice = wt[1:5:2] → [2,4]
```


Rimozione elementi da una lista

- Python offre diverse alternative
  - *pop()*
  - *remove()*
  - *del*
- Differenze
  - lista.pop(ind1): rimuove l'elemento di *indice ind1 e lo ritorna*
  - lista.remove(elem1): rimuove l'*elemento elem1 (matching) senza ritornarlo*
  - del lista[ind1]: *statement* che rimuove l'elemento di *indice ind1*
    - del lista[-2:]: rimuove gli *ultimi due elementi di una lista (uso di operatore di slicing)*

Copia di una lista

Come creare una copia di una lista?

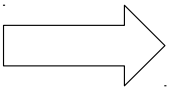
```
>>> a = [1,2,3]
>>> b = a
>>> print(b)
[1, 2, 3]
>>> b[1] = 10
>>> print(b)
[1, 10, 3]
>>> print(a)
[1, 10, 3]
```

E' stato modificato anche l'originale!!

Copia di una lista

Come creare una copia di una lista?

```
>>> a = [1,2,3]
>>> b = a
>>> print(b)
[1, 2, 3]
>>> b[1] = 10
>>> print(b)
[1, 10, 3]
>>> print(a)
[1, 10, 3]
```



```
>>> a = [1,2,3]
>>> b = a[:]
>>> print(b)
[1, 2, 3]
>>> b[1] = 10
>>> print(b)
[1, 10, 3]
>>> print(a)
[1, 2, 3]
```

E' stato modificato anche l'originale!!

→ESEMPI
liste.py

Metodi definiti sulle liste

- **Funzione len(lista): ritorna il numero di elementi contenuti in una lista**
- **Metodo lista.sort(): ordina gli oggetti contenuti - modifica lista in-place!**

>>> lista = [2, 6, 4, 10, 1, 3, 9, 5, 7, 8]

>>> lista.sort() ; print(lista)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

- **sorted(lista) non modifica la lista originale**
- **Operatore in: ricerca elemento in una lista**

  >>> 6 in lista → True

→ESERCIZIO
esercizio02.py

Tuple

- **Una tupla in Python è un sequenza di oggetti eterogenei esattamente come le liste, con la differenza che le tuple non possono essere modificate**
  - ***Sono immutabili (come le stringhe)***
- **Nella definizione utilizzano parentesi tonde**

>>> tup1 = ('one', 'two', 12, 25)

>>> tup2 = (3, 5, 6, 9)

>>> print(tup1[0], tup2[1:3])

one (5, 6)

Tipo: stringa

Tipo: tupla
type(tup2[1:3])

Vantaggi:
essendo immutabili,
potremo usarle
come chiave nei **dictionary**

Set

- In Python il set implementa il concetto di insieme come collezione non ordinata (no accesso tramite indice) di oggetti senza duplicati
- Spesso usati per rimuovere velocemente duplicati o per efficienti test di appartenenza ad un insieme
- Creato con set(oggetto_iterabile)

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']   #caso creazione da lista con duplicati
>>> fruit = set(basket)     # crea un set di nome fruit
>>> fruit                   # senza duplicati
set(['orange', 'pear', 'apple', 'banana'])
>>> fruit.add('kiwi') ; fruit.discard('apple'); fruit
set(['orange', 'kiwi', 'pear', 'banana']) #non ordinato
```


Metodi a disposizione sui set

- Alcuni metodi a disposizione dei set
  - Cardinalità del set S: len(S)
  - Appartenenza all'insieme: x in S, x not in S
  - Disgiunzione: S1.isdisjoint(S2)
  - Unione: S1.union(S2) (simbolo “|”)
  - Intersezione: S1.intersection(S2) (simbolo “&”)
  - Differenza: S1.difference(S2) (simbolo “-”)

Set: esempio

```
>>> a = set('abracadabra') #creato da stringa
>>> b = set('alacazam')
>>> a                          # eliminazione duplicati
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                      # lettere in a ma non in b
set(['r', 'd', 'b'])
>>> a | b                      # lettere in a o b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                     # lettere in entrambi
>>> set(['a', 'c'])
```


Dictionary

- In Python un dictionary rappresenta l'astrazione di una lista associativa di coppie chiave-valore
  - I valori possono essere oggetti di qualunque tipo (dynamic typing)
  - Le chiavi possono essere: numeri, stringhe o tuple (oggetti non modificabili)
- Definizione di dictionary (due alternative):
  - Sequenza di coppie key: value separate da virgola e confinate in parentesi graffe
  - Uso del costruttore dict(key=value)

Definizioni di dictionary

```
>>> wt = { 'one': 1, 'two': 2 }
>>> wt = dict(one=1, two=2)
```

Definizioni equivalenti

```
>>> wt = { }  #dictionary vuoto
>>> wt.update({'three':3})
```

Aggiunta o modifica elementi

Indicizzazione attraverso le chiavi (uniche in un dictionary)

```
>>> print(wt['one'])  → 1
>>> wt['four'] = 4  → aggiunge elemento se manca
```


Accesso elementi di un dictionary

- **Più usati: dictionary uno-a-uno (es. precedenti)**
- **Inoltre: dictionary uno-a-molti (oggetti lista come valori)**

```
>>> lett = { 'vocali': ['a','e','i','o','u'], \
      'consonanti': ['b','c','d','f'] }
```

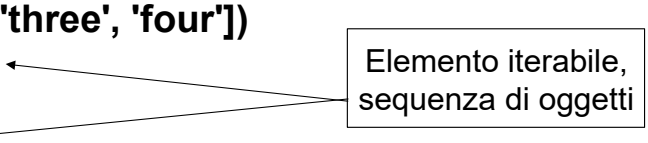
- **Metodi definiti sui dictionary:**
  - **wt.keys(): restituisce un *iteratore* su tutte le chiavi**
  - **wt.values(): restituisce un *iteratore* su tutti i valori**
  - **wt.items(): restituisce un un *iteratore* su tutte le *tuple (chiave, valore) del dictionary***

    ('one', 1), ('two', 2), ('three', 3), ('four', 4)

Liste da .keys(), .values(), .items()

Per ottenere una lista dai metodi di accesso agli elementi del dictionary → funzione/costruttore list()

```
>>> wt.keys()
dict_keys(['one', 'two', 'three', 'four'])
>>> type(wt.keys())
<class 'dict_keys'>
>>> list(wt.keys())
['one', 'two', 'three', 'four']
>>> list(wt.values())
[1, 2, 3, 4]
>>> list(wt.items())
[('one', 1), ('two', 2), ('three', 3), ('four', 4)]
```



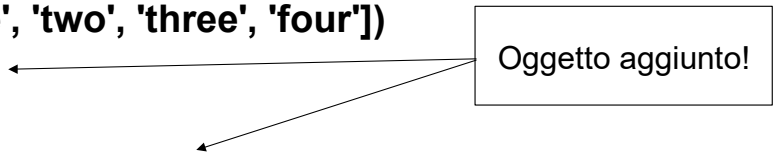
View objects

wt.keys(), wt.values(), wt.items()

Si tratta di *view objects*: forniscono una *vista dinamica* sul contenuto del dizionario

- **Quando il dizionario cambia, un view object riflette il cambiamento**

```
>>> wt={'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> wobj1=wt.keys()
>>> wobj1
dict_keys(['one', 'two', 'three', 'four'])
>>> wt['five']=5
>>> wobj1
dict_keys(['one', 'two', 'five', 'three', 'four'])
```



Eliminazione e ricerca chiavi

- **Uso di del o pop() per eliminare una chiave ed il suo valore associato – accesso tramite chiave**

del wt['one'] *equivale a* **wt.pop('one')**

- **Controllo di presenza (ricerca): operatore in**
  - **key in dictionary: ritorna True se il dictionary ha la chiave key, False altrimenti**
  - **'one' in wt → True**

NOTA: l'operatore in è quello già visto su stringhe, liste e set

Tuple come chiavi d'accesso

- **Le tuple possono essere usate come chiavi per un dictionary in quanto oggetti immutabili**

```
Es.
t1=('Mario', 'Rossi')
t2=('Carlo', 'Bellini')
d = { }
d[t1] = 4218
d[t2] = 5617
print(d)
```

→ {('Mario','Rossi'): 4218, ('Carlo', 'Bellini'): 5617}

Costrutti del Python

Costrutto condizionale: if-elif-else

```
if condizione:
    statement
elif condizione:
    statement
else:
    statement
```

Keyword *elif*

Nota: negazione con *if not*

Costrutti iterativi: while e for

```
while condizione:
   statement
```

Identico al C

```
for variabile in elenco_oggetti:
   statement
```

Forma molto sintetica
Uso dell'operatore in

Comandi di salto

→ESEMPI
while_else.py

- Comando break:
  - Interrompe un ciclo for/while
- Comando continue:
  - Salta all'iterazione for/while successiva
- Clausola else:
  - Può essere inserita alla fine di un blocco relativo ad un ciclo
  - Viene eseguita (una volta sola) se un ciclo termina le sue iterazioni a seguito della valutazione False della condizione del ciclo
  - Non viene eseguita in caso di break

Ciclo for

for variabile in elenco_oggetti:

statement

- ***Tipico uso*: iterare su una sequenza di numeri interi consecutivi di lunghezza determinata**

  Es. contatore usato come indice per la scansione di sequenza *seq* di elementi di lunghezza N

```
>>> for i in [0, 1, 2, …, N-1]:
…       print(seq[i])
```

"…" Shell interattiva Prompt di secondo livello (indentazione)

- **Funzione range(): implementa il concetto di sequenza non modificabile - fornisce un oggetto utilizzabile per le iterazioni, e anche accessibile con indice**

```
>>> for i in range(N):
…       print(seq[i])
```

```
>>> type(range(10))
<class 'range'>
```

```
>>> a=range(10)
>>> a[6]
6
```


For e range

Uso di range(): numero di parametri variabili

	Indice i assume i valori:
for i in range(10): →	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
for i in range(1,10) →	1, 2, 3, 4, 5, 6, 7, 8, 9
for i in range(1,10,2) →	1, 3, 5, 7, 9

Iterazione su una lista *xlist*

```
>>> for i in range(len(xlist)):
…       print(xlist[i])
```

Riutilizzo in più cicli for successivi

```
>>> range_xlist = range(len(xlist))
>>> for i in range_xlist:
…       print(xlist[i])
```

→ESEMPI
range.py

Esempio uso range

Stampare il numero di lettere associate ad ogni parola contenuta nella stringa:

stringa = "Trattasi di una stringa di prova per il corso LD"

Effettuare anche la stampa a rovescio (dall'ultima parola alla prima)

Creare liste numeriche con range

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]

>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

Funzione *list()*: crea una lista a partire da un **oggetto iterabile**

Applicazione ciclo for

- **Oltre all'uso con range(), in Python il ciclo for può iterare su tantissimi tipi di oggetti**
  - ***Con lo stesso identico statement!***

Liste:

```
>>> obj = [1,3,10,0]
>>> for i in obj:
…       print(i)
1
3
10
0
```

Stringhe:

```
>>> obj = "Ciao"
>>> for i in obj:
…       print(i)
C
i
a
o
```

Tuple:

```
>>> obj = (2,'a',5,'p')
>>> for i in obj:
…       print(i)
2
a
5
p
```


Applicazione ciclo for

→ESERCIZI
esercizio03.py
esercizio04.py

Oltre all'uso con range(), in Python il ciclo for può iterare su tantissimi tipi di oggetti

- ***Con lo stesso identico statement!***

Set:

```
>>> obj = set('set')
>>> for i in obj:
…       print(i)
s
e
t
```

Dictionary:

```
>>> obj = dict(one=1, two=2, three=3)
>>> for i in obj:
…       print(i, obj[i])
three 3
two 2
one 1
```

L'operatore in itera sulle *chiavi* del dictionary

E sarà possibile iterare anche sui file...

Esercizi

- *Esercizio esercizio03.py*
  - *Range, Dictionary, applicazione di costrutto if e di ciclo for*
- *Esercizio esercizio04.py*
  - *Dictionary, manipolazione di stringhe*

Ciclo for e funzione enumerate

- **Funzione enumerate(sequence, start=0): ritorna coppie del tipo (indice, valore) utilizzabili in un ciclo for con doppio assegnamento**
  - **Sequence deve essere un oggetto iterabile (es. lista, tuple, set, ...)**

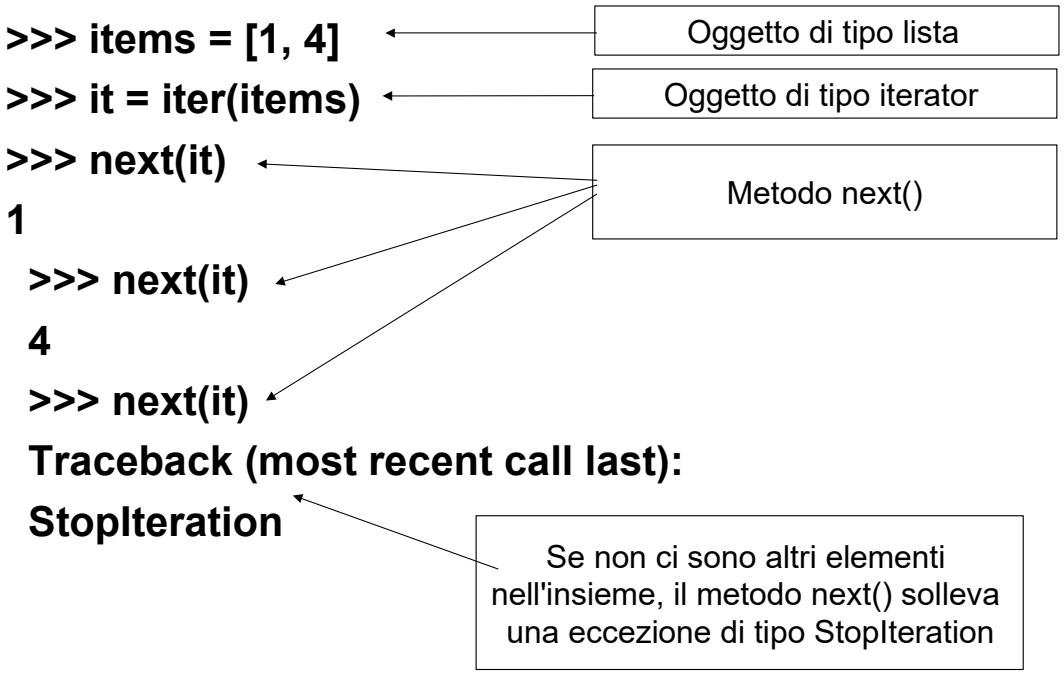
```
for index, item in enumerate(sequence):
    print index, item
>>> lista = [2, “ciao”, 4]
>>> for index, item in enumerate(lista):
...         print("lista[" + str(index) + "]: ", item)
lista[0]:  2
lista[1]:  ciao
lista[2]:  4
```

Oggetti iterator

- **Qual'è il meccanismo che ci permette di iterare su oggetti così eterogenei usando sempre lo stesso operatore in?**
- **Gli oggetti iterator implementano il concetto di iterazione su un *insieme* di elementi diversi**
  - **E' possibile iterare su qualunque *insieme* di elementi che supporta il metodo *iter(insieme)*, che restituisce un oggetto di tipo iterator**
  - **Un oggetto iterator supporta un metodo *next(iterator)* che ritorna l'elemento successivo dell'insieme**
- **Ogni oggetto (insieme di elementi) che supporta in questo modo i metodi iter() e next() è iterabile: liste, stringhe, tuple, set, dictionary, file**

Meccanismo di iterazione

→ESEMPI
iterator.py



Esempi

Diverse funzioni Python richiedono come parametri in ingresso oggetti iterabili → utilizzano a basso livello il concetto di iterator

Es:

- **sorted(iterable) → ritorna una lista ordinata degli elementi di iterable**
- **sum(iterable) → ritorna la somma degli elementi di iterable (che si assumono numeri)**
- **max(iterable) → ritorna il massimo degli elementi**
- **list(iterable) → ritorna una lista composta degli elementi di iterable**
- **x in iterable → ritorna True o False se x è presente in iterable**

Esempi d'uso

```
sum([1,2,3,4]) ↔ sum(range(1,5))
wt={'ten': 10, 'one': 1, 'four': 4, 'three': 3}
>>> sorted(wt)
['four', 'one', 'ten', 'three']
>>> sorted(wt.keys())
['four', 'one', 'ten', 'three']
>>> sorted(wt.values())
[1, 3, 4, 10]
>>> sorted(wt.items())
[('four', 4), ('one', 1), ('ten', 10), ('three', 3)]
>>> sum(wt.values())
18
```

Esempi

Alcune funzioni Python restituiscono come valore di ritorno degli iteratori

Es:

- **reversed(sequence) → ritorna un interatore che scorre la sequence al contrario**
  - **Sequence: sequenza ordinata di elementi (stringhe, liste, tuple)**

```
>>> x=['Claudia', 'Anna', 'Dario', 'Barbara']
>>> type(reversed(x))
<class 'list_reverseiterator'>
>>> for i in reversed(sorted(x)):
…             print(i)
```

Stampa i nomi in ordine alfabetico inverso