Le stringhe in Python

In Python una stringa è una sequenza di caratteri racchiusa tra apici e ordinata in base a un indice che parte dal valore 0.

La funzione len

La funzione `len` restituisce il numero di caratteri contenuti in una stringa.

Esempio (1)

```
nome='Pacinotti'
print(len(nome))
```

Viene stampato il numero di caratteri di 'Pacinotti': 9.

L'operatore in

La parola `in` è un operatore booleano che confronta due stringhe e restituisce True se la prima è una sottostringa della seconda, altrimenti restituisce False.

Esempio (2)

```
print('a' in 'Pacinotti')
```

stampa True perché la lettera a è contenuta in "Pacinotti".

```
print('b' in 'Pacinotti')
```

stampa False perché la lettera b non è contenuta in "Pacinotti".

Esempio (3)

Date due parole inserite dall'utente, la funzione `in_entrambe` stampa tutte le lettere che compaiono nelle due parole.

```
parola1=input('Scrivi la prima parola ')
parola2=input('Scrivi la seconda parola ')
def in_entrambe(parola1, parola2):
      for lettera in parola1:
            if lettera in parola2:
                  print(lettera)
in_entrambe(parola1, parola2)
```

Operatore [] per selezione dei caratteri

Si può accedere al singolo carattere di una stringa usando l'operatore parentesi quadra: `nome_stringa[5]` seleziona il carattere della stringa nella posizione 5.

Ricordare che le posizioni dei caratteri partono da 0; nella stringa 'Pacinotti', per esempio:

P è il carattere nella posizione 0;

a è il carattere nella posizione 1;

c è il carattere nella posizione 2.

Si possono usare numeri interi negativi per contare a ritroso dall'ultima lettera:

i, l'ultima lettera, è nella posizione -1;

t, la penultima lettera, è nella posizione -2;

P, la lettera iniziale, contando a ritroso è nella posizione -9.

Esempio (4)

```
nome='Pacinotti'      #definisce la variabile nome
lettera=nome[0]       #seleziona il carattere nella posizione 0 e lo assegna
                        alla variabile lettera
print(lettera)        #stampa sullo schermo la variabile lettera
```

Il carattere stampato sullo schermo è la lettera P.

Esempio (5)

```
nome='Pacinotti'      #definisce la variabile nome
lettera=nome[-9]      #seleziona il carattere nella posizione 9 partendo da
                        destra e lo assegna alla variabile lettera
print(lettera)        #stampa sullo schermo la variabile lettera
```

Il carattere stampato sullo schermo è ancora la lettera P.

Esempio (6)

```
nome='Pacinotti'
i=1                       #definisce la variabile i con valore 1
print(nome[i])            #stampa il carattere nella posizione i=1
print(nome[i+1]           #stampa il carattere nella posizione i+1=2
print(nome[i+0.5])        #in questo caso la posizione non è un numero intero: viene stampato
                           un messaggio d'errore
```

Si potrebbe usare, per selezionare l'ultimo carattere di una stringa, invece della posizione negativa -1, la posizione data dalla lunghezza della stringa meno uno.

Esempio (7)

```
nome='Pacinotti'
lunghezza=len(nome)
print(nome[lunghezza-1])      #stampa il carattere nell'ultima posizione
```

Attraversamento di stringa con ciclo for

In diversi casi si richiede l'elaborazione di una stringa, un carattere per volta: questo tipo di elaborazione è detto attraversamento di una stringa.

Possiamo usare due modi per attraversare una stringa, con while o con for.

Con ciclo while:

```
nome='Pacinotti'
i=0                           #definisce la variabile indice i ponendola
                               inizialmente a zero
while i<len(nome):            #comincia il ciclo while: sino a quando
                               l'indice i assume un valore inferiore alla lunghezza della stringa…
      lettera=nome[i]         #...alla variabile lettera viene associato il carattere nella posizione i
      print(lettera)          #viene stampata su schermo la lettera
      i=i+1                   #viene aumentato di 1 il valore dell'indice i
```

La condizione del ciclo è che l'indice sia inferiore alla lunghezza della stringa; quando i diventa uguale alla lunghezza della stringa, la condizione diventa falsa e il corpo del ciclo non viene più eseguito.

Con ciclo for:

```
nome='Pacinotti'
for lettera in nome:
      print(lettera)
```

A ogni ciclo, il successivo carattere della stringa viene assegnato alla variabile lettera.

Operazioni sulle stringhe

L'operatore + esegue il concatenamento tra due stringhe: unisce le stringhe collegandole ai due estremi.

Esempio (8)

```
nome1='Liceo '
nome2='scientifico '
nome3='Antonio '
nome4='Pacinotti'
print(nome1+nome2+nome3+nome4)
```

Viene stampato 'Liceo scientifico Antonio Pacinotti'.

L'operatore * esegue la ripetizione di una stringa.

Esempio (9)

```
nome='Pacinotti '
print(nome*100)
```

Viene visualizzata 100 volte la stringa 'Pacinotti '.

Vediamo come si può usare il concatenamento e un ciclo per generare una serie disposta in ordine alfabetico.

Esempio (10)

```
prefissi='JKLMNOPQ'
suffisso='ack'
for lettera in prefissi:
    print(lettera + suffisso)
```

Il risultato del programma è la stampa, su righe diverse, dei nomi Jack, Kack, Lack, Mack, Nack, Oack, Pack, Qack.

Elaborazione tramite codice ASCII

Come tutti i linguaggi di programmazione, Python memorizza i caratteri alfanumerici tramite il rispettivo codice numerico della tabella ASCII.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Su questo fatto si basano numerose operazioni con cui si può elaborare il contenuto delle stringhe.

Esempio (11)

```
nome1=”Mario”
nome2=”Maria”
if nome1>nome2:
      print(nome1)
else:
      print(nome2)
```

La lettera o ha codice ASCII 111, la lettera a ha codice ASCII 97: per tale motivo la stringa ‘Maria’ risulta minore di ‘Mario’.

Lo stesso principio consente di controllare la correttezza di una password inserita dall'utente.

Esempio (12)

```
pwd='PAc1n8'        #definiamo la stringa 'PAc1n8'e la chiamiamo pwd
ins=input("Inserisci la password")  #definiamo la variabile ins immessa
                                    dall'utente
if ins!=pwd:        #se la password inserita dall'utente è diversa da pwd
      print("Password errata, accesso non consentito.")  #viene stampato
                                                         il messaggio di errore
else:               #altrimenti, se la password inserita è uguale a pwd…
      print("Password corretta, procedi.")  #...viene stampato questo messaggio
```

Anche le cifre sono lette come caratteri e hanno un corrispondente numero ASCII.

La funzione input interpreta come stringhe i caratteri inseriti.

La funzione `ord()` restituisce il codice ASCII del carattere dato come argomento.

Con `print(ord('1'))` viene stampato 49, il codice ASCII di 1.

Esempio (13)

Con il seguente programma vengono stampate tutte le lettere, con il rispettivo codice ASCII:

```
pwd='PAc1n8'
for let in pwd:
      print(let, ord(let))
```

La funzione "inversa" di `ord()` è `chr()`, che restituisce un carattere se come argomento le si passa il corrispondente codice numerico ASCII.

`Con print(chr(64))` il risultato è @, corrispondente al codice numerico ASCII 64.

Slicing di stringa

Con lo slicing (dall'inglese to slice, "affettare") si selezionano parti di stringa.

```
nome_stringa[start:stop:step]
```

I tre argomenti tra le parentesi quadre possono essere numeri interi espliciti oppure variabili che hanno valori numerici interi:

start è l'indice del carattere da cui iniziare la selezione;

stop è l'indice esterno al termine della selezione (l'ultimo carattere selezionato avrà l'indice stop-1);

step è l'intervallo tra un carattere selezionato e il successivo ed è un argomento facoltativo: se si omette, lo slicing userà il valore di default step=1.

Esempio (14)

```
stringa="PACINOTTI"
#slicing che seleziona i caratteri con indice pari
print(stringa[0:9:2])
#slicing che seleziona i caratteri con indice dispari
print(stringa[1:9:2])
#slicing che seleziona "CINO"
print(stringa[2:6])
#slicing che seleziona l'intera stringa, con omissione del parametro stop
print(stringa[0:])
#slicing che seleziona i primi quattro caratteri, con omissione del parametro start
print(stringa[:4])
#non digitando i due punti, si seleziona solo il carattere nella posizione 4, coerentemente con quanto visto in precedenza
print(stringa[4])
```

Nelle righe 7, 9, 11 si omette step: per default il valore è 1.

Nelle righe 9 e 11 si omettono rispettivamente il parametro stop e il parametro start; i due punti devono però essere mantenuti (in caso contrario si tornerebbe alla selezione di un singolo carattere della stringa, come nella riga 13).

Ricerca di ricorrenze e sottostringhe

Spesso si vuole verificare la presenza di un carattere all'interno di una stringa o calcolare quante volte un dato carattere ricorra nella stringa.

Esempio (15): ricerca di un carattere

```
frase=input('Inserisci una frase:    ')
carattere=input('Quale carattere vuoi cercare?    ')
def trova(frase, carattere): #inizia la definizione della funzione trova
      i=0                          #indice i posto inizialmente a zero
      while i<len(frase):
            if frase[i]==carattere: #confronta il carattere della stringa in posizione i con il
                                      carattere cercato
                  return i          #se trova il carattere, termina il ciclo e restituisce il valore
                                      dell'indice
            i+=1                    #se non trova il carattere, aumenta di 1 l'indice e rieffettua
                                      il confronto tra carattere nella stringa e carattere cercato
      return -1                  #se non trova il carattere, restituisce -1
print(trova(frase, carattere)) #stampa il valore della funzione
```

In tale esempio viene quindi stampata la posizione della prima volta in cui compare il carattere cercato oppure, se il carattere non compare nella stringa, viene stampato -1.

Esempio (16): contatore

```
frase=input('Inserisci una frase:  ')
carattere_cercato=input('Quale carattere vuoi cercare?  ')
conta=0
for let in frase:
      if let==carattere_cercato:
            conta+=1
print('Il carattere', cercato, 'ricorre', conta, 'volte.')
```

Nella riga 3 viene dichiarata la variabile `conta`, che servirà per contare le ricorrenze, e viene inizializzata a zero.
Nella riga 4 inizia il ciclo `for` che itera su tutti i caratteri della frase e contiene una struttura condizionale: ogni volta che il carattere corrisponde a quello cercato, si incrementa di una unità il valore di `conta`.

Vogliamo ora verificare se una stringa sia contenuta all'interno di un'altra (tipo di ricerca effettuato dai motori di ricerca nel web).

Si utilizza la parola chiave `in`.

Esempio (17)

```
frase='''In matematica, il logaritmo di un numero in una data base è l'esponente al quale la base deve essere elevata per ottenere il numero stesso.'''
str_cercata=input('Quale stringa vuoi cercare?  ')
if cercata in frase:
     print('La stringa',str_cercata,'è presente nella frase.')
else:
     print('La stringa',str_cercata,'non è presente nella frase.')
```

Le stringhe in Python non sono modificabili!

Il contenuto di una stringa memorizzata in Python non si può alterare: per avere una stringa diversa, bisogna crearne una nuova!

Esempio (18)

Cerchiamo di modificare la stringa da Mario a Maria.

```
nome='Maria'
nome[4]='o'    #assegna alla posizione 4 della stringa il carattere "o"
print(nome)
```

Avviando il programma, viene dato un avviso di errore:

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

Vediamo ora, nell'esempio 19, un programma che confronta un PIN inserito dall'utente con un valore memorizzato: dapprima verifica che il PIN inserito contenga solo caratteri numerici, poi verifica la correttezza del codice inserito.

Esempio (19)

```
PIN_impostato='4321'
PIN=input('Inserisci il PIN: ')  #acquisice il PIN dall'utente, che sarà di tipo stringa
ver=True    #inizializza la variabile booleana ver per controllare che il PIN inserito sia numerico
indice = 0    # inizializza a zero la variabile indice che rappresenta l'indice dei caratteri immessi
            dall'utente (PIN[0], PIN[1] etc.)

while indice<len(PIN) and ver==True:    # introduce un ciclo di tipo while che itera
                                        sui caratteri inseriti sino a quando il valore di indice
                                        risulta minore della lunghezza della stringa e ver rimane True

      if ord(PIN[indice])>47 and ord(PIN[indice])<58:
            indice+=1
      else:
            ver=False                   # controlla che ogni carattere sia di tipo numerico, facendo
                                        riferimento al codice ASCII del carattere controllato;
                                        se si incontra un carattere non numerico, ver diventa False e il ciclo
                                        si interrompe
if ver==False:
      print('Attenzione: devi digitare soltanto cifre!')
else:
      if PIN!=PIN_impostato:        #controlla la correttezza del PIN  inserito dall'utente
            print('Il PIN inserito non è corretto')
      else:
            print('Bene, il PIN inserito è corretto!')
```

Metodi delle stringhe

Oltre a essere un particolare tipo di dato che contiene una sequenza indicizzata di caratteri alfanumerici, in Python le stringhe sono anche oggetti.

L'oggetto, nella terminologia dell'OOP (Programmazione orientata agli oggetti), è una struttura del linguaggio che contiene sia "dati" (anche di tipo diverso tra loro) sia "metodi", ossia funzioni proprie.

Usando un metodo particolare degli oggetti di Python si può ottenere un codice più compatto.

Per applicare un metodo a un oggetto si usa l'operatore punto tra il nome della variabile e il nome del metodo:

nome_stringa.nome_metodo(*argomenti*)

Ad esempio, data una stringa chiamata nome:

len(nome) è un esempio di funzione generale del linguaggio applicata alla stringa;

nome.len() è un esempio di metodo dell'oggetto stringa.

La forma di notazione a punto è detta "dot notation" in inglese. Le parentesi vuote indicano che il metodo non ha argomenti. La chiamata di un metodo è detta invocazione.

Si può utilizzare il metodo **`isdigit()`** per verificare la natura dei dati contenuti in una stringa: restituisce il valore True se i dati sono soltanto numerici, False nel caso contrario. Con l'uso di tale metodo non è più necessario dover iterare con un ciclo sui singoli caratteri della stringa.

Esempio (19 bis)

```
PIN_impostato='4321'
PIN=input('Inserisci il PIN: ')
if PIN.isdigit():
      if PIN!=PIN_impostato:
            print('Attenzione, il PIN non è corretto!')
      else:
            print('Il PIN è corretto.')
else:
      print('Devi introdurre solo caratteri numerici!')
```

Nella tabella sono riportati alcuni metodi di stringa.

Metodo	Descrizione
`isalnum()`	restituisce True se la stringa non è vuota e contiene soltanto caratteri alfanumerici, altrimenti restituisce False
`isalpha()`	restituisce True se la stringa non è vuota e contiene soltanto caratteri alfabetici, altrimenti restituisce False
`isdigit()`	restituisce True se la stringa non è vuota e contiene soltanto caratteri numerici, altrimenti restituisce False
`islower()`	restituisce True se la stringa contiene almeno un carattere alfabetico e tutti i caratteri alfabetici sono minuscoli, altrimenti restituisce False
`isupper()`	restituisce True se la stringa contiene almeno un carattere alfabetico e tutti i caratteri alfabetici sono maiuscoli, altrimenti restituisce False

Creare stringhe a partire da altre stringhe

Non si può modificare il contenuto di una stringa, ma esistono metodi che consentono di creare nuove stringhe a partire da altre stringhe.

Esempio (20)

Trasformiamo tutte le lettere di una stringa in maiuscole .

```
nome_inserito=input('Inserisci il tuo nome: ')
nome_maiuscolo=nome_inserito.upper()
print(nome_maiuscolo)
```

Esempio (21)

Trasformiamo tutte le lettere di una stringa in minuscole.

```
nome_inserito=input('Inserisci il tuo nome: ')
nome_minuscolo=nome_inserito.lower()
print(nome_minuscolo)
```

oppure, senza definire una nuova variabile:

```
nome_inserito=input('Inserisci il tuo nome: ')
print(nome_inserito.lower())
```

Con il metodo **`replace()`** si può creare una nuova stringa modificando al volo il contenuto di una stringa esistente. Si può creare una stringa di lunghezza diversa da quella originaria.

`replace()` ha due argomenti:

- sottostringa da sostituire;

- sottostringa che comparirà al suo posto, ovunque appaia la sottostringa sostituita.

Ricordare che Python è sensibile alle maiuscole!

Esempio (22)

```
stringa='Asso'
print(stringa)
stringa2=stringa.replace('A', 'o')
print(stringa2)
stringa3=stringa2.replace('s','t')
print(stringa3)
stringa4=stringa.replace('As', 'sas')
print(stringa4)
stringa5=stringa4.replace('s', 't')
print(stringa5)
```

`find()` e **`index()`** individuano in una stringa la prima occorrenza della sottostringa passata come argomento e restituiscono il valore dell'indice a cui inizia.

C'è una grande differenza tra i due metodi: se la sottostringa cercata non è presente, `find()` restituisce il valore -1, mentre `index()` segnala un errore e blocca il programma.

Esempio (22)

```
stringa=('Liceo scientifico Antonio Pacinotti')
print('Ricerca di scientifico con find:',stringa.find('scientifico'))
print('Ricerca di classico con find:',stringa.find('classico'))
print('Ricerca di scientifico con index:', stringa.index('scientifico'))
print('Ricerca di classico con index:', stringa.index('classico'))
```

I metodi **`rfind()`** e **`rindex()`** restituiscono l'indice corrispondente all'ultima ricorrenza di una sottostringa in una stringa.

Il metodo **`count()`** consente di effettuare il conteggio delle ricorrenze per ogni sottostringa.

Esempio (23)

```
stringa=('Liceo scientifico Antonio Pacinotti')
Print(stringa)
print('La sottostringa ti ricorre',stringa.count('ti'), 'volte.')
print('La lettera a ricorre',stringa.count('a'), 'volte.')
#Ricordare che Python è sensibile alle maiuscole!
print('La lettera i ricorre',stringa.count('i'), 'volte.')
```