A

Linguaggio Python: operatori

Gli operatori del linguaggio Python sono elencati nella tabella seguente, in gruppi di precedenza decrescente. Le righe orizzontali all'interno della tabella delimitano un gruppo e segnalano un cambiamento nella precedenza tra gli operatori. Gli operatori che hanno una precedenza più elevata si abbinano ai propri operandi in modo prioritario rispetto a quelli che hanno una precedenza minore. Ad esempio, `x + y * z` significa `x + (y * z)`, perché l'operatore `*` ha una precedenza più elevata dell'operatore `+`. Sempre osservando la tabella, si può capire che `x and y or z` significa `(x and y) or z`, perché l'operatore `or` ha un livello di precedenza inferiore.

L'*associatività* di un operatore indica se lo si deve associare da sinistra a destra oppure da destra a sinistra. In Python, tutti gli operatori si associano da sinistra a destra, tranne l'operatore di elevamento a potenza. Ad esempio, `x - y - z` significa `(x - y) - z`, perché l'operatore `-` si associa da sinistra a destra. L'operatore di elevamento a potenza, invece, si associa da destra a sinistra, quindi `x ** y ** z` significa `x ** (y ** z)`.

Operatore	Descrizione	Dove
`[]`	Indice	Paragrafo 2.4.4, Paragrafo 6.1.2
`[:]`	Porzione	Argomenti avanzati 6.3
`()`	Invocazione di funzione o di metodo	Paragrafo 2.2.4
`.`	Accesso a variabile di esemplare o a metodo	Paragrafo 2.4.5, Paragrafo 9.2
`**`	Elevamento a potenza	Paragrafo 2.2.2

(continua)

(segue)

Operatore	Descrizione	Dove
`+` *(unario)*	Positivo	Paragrafo 2.2.1
`-` *(unario)*	Negativo	Paragrafo 2.2.1
`~` *(unario)*	Not bit a bit	Appendice E
`*`	Moltiplicazione	Paragrafo 2.2.1
`/`	Divisione in campo reale	Paragrafo 2.2.1
`//`	Divisione intera	Paragrafo 2.2.3
`%`	Resto	Paragrafo 2.2.3
`*`	Replicazione di sequenza	Paragrafo 2.4.2
`+`	Addizione	Paragrafo 2.2.1
`+`	Concatenazione di sequenza	Paragrafo 2.4.2
`-`	Sottrazione	Paragrafo 2.2.1
`<<`	Scorrimento a sinistra bit a bit	Appendice E
`>>`	Scorrimento a destra bit a bit	Appendice E
`&`	And bit a bit	Appendice E
`^`	Or esclusivo bit a bit	Appendice E
`¦`	Or bit a bit	Appendice E
`in`	Verifica di appartenenza a un contenitore	Paragrafo 3.8, Paragrafo 6.3.5
`not in`	Verifica di non appartenenza a un contenitore	Paragrafo 3.8, Paragrafo 6.3.5
`is`	Verifica di identità: è un alias	Paragrafo 9.10.1
`is not`	Verifica di identità: non è un alias	Paragrafo 9.10.1
`<`	Minore di	Paragrafo 3.2
`<=`	Minore di o uguale a	Paragrafo 3.2
`>`	Maggiore di	Paragrafo 3.2
`>=`	Maggiore di o uguale a	Paragrafo 3.2
`!=`	Diverso da	Paragrafo 3.2
`==`	Uguale a	Paragrafo 3.2
`not`	Not booleano	Paragrafo 3.7
`and`	And booleano	Paragrafo 3.7
`or`	Or booleano	Paragrafo 3.7
`if ... else`	Espressione condizionale	Argomenti avanzati 3.1
`lambda`	Funzione anonima	Argomento non trattato

B

Linguaggio Python: parole riservate

Parola riservata	Descrizione	Dove
`and`	And booleano	Paragrafo 3.7
`as`	Usata come parte di un `try` o con una clausola `with`, per specificare un nome alternativo per un oggetto	Paragrafo 7.5
`assert`	Un'asserzione relativa a una condizione che deve essere soddisfatta	Argomento non trattato
`break`	Interrompe il ciclo in esecuzione	Argomenti avanzati 4.2
`class`	Definisce una classe	Paragrafo 9.2
`continue`	Ignora la parte restante del corpo di un ciclo	Argomento non trattato
`def`	Definisce una funzione o un metodo	Paragrafo 5.2, Paragrafo 9.2
`del`	Elimina un elemento da un contenitore	Argomento non trattato
`elif`	Una diramazione condizionale alternativa	Paragrafo 3.4
`else`	La clausola alternativa di un enunciato `if`	Paragrafo 3.1
`except`	Il gestore di un'eccezione in un blocco `try`	Paragrafo 7.5
`finally`	Una clausola di un blocco `try` che viene sempre eseguita	Paragrafo 7.5
`for`	Un ciclo per scandire gli elementi di un contenitore	Paragrafo 4.6
`False`	Il valore booleano "falso"	Paragrafo 3.7

(continua)

(segue)

Parola riservata	Descrizione	Dove
`from`	Usata con l'enunciato `import` per includere elementi di un modulo	Paragrafo 2.2
`global`	Dichiara che una variabile ha visibilità globale	Paragrafo 5.8
`if`	Una diramazione condizionale	Paragrafo 3.1
`import`	Include in un modulo singoli elementi o l'intero contenuto di un altro modulo	Paragrafo 2.2, Argomenti avanzati 2.1
`in`	Verifica l'appartenenza a un contenitore	Paragrafo 3.8
`is`	Verifica se una variabile è un alias	Paragrafo 9.10
`lambda`	Crea una funzione anonima	Argomento non trattato
`None`	Un valore speciale che indica un riferimento non esistente	Paragrafo 9.10
`Not`	Not booleano	Paragrafo 3.7
`or`	Or booleano	Paragrafo 3.7
`pass`	Un segnaposto, da inserire dove è richiesto un enunciato	Argomento non trattato
`raise`	Solleva un'eccezione	Paragrafo 7.5
`return`	Termina l'esecuzione di un metodo ed eventualmente restituisce un valore	Paragrafo 5.4
`True`	Il valore booleano "vero"	Paragrafo 3.7
`try`	Un blocco di codice con gestori di eccezioni o un gestore `finally`	Paragrafo 7.5
`with`	Un blocco di codice che viene eseguito all'interno di un contesto specifico	Argomenti avanzati 7.4
`while`	Un ciclo	Paragrafo 4.1
`yield`	Restituisce il risultato di una funzione generatrice	Argomento non trattato

C

Linguaggio Python: libreria standard

Questa appendice descrive sinteticamente le principali funzioni e classi della libreria standard del linguaggio Python, oltre al modulo grafico usato in questo libro.

Funzioni predefinite

- **`abs(`*x*`)`**
  Questa funzione calcola il valore assoluto di *x*, cioè $|x|$.
  Parametro: *x*, un valore numerico
  Restituisce: il valore assoluto dell'argomento

- **`bytes()`**
- **`bytes(`*x*`)`**
  Questa funzione crea una nuova sequenza di `bytes`. Se non viene fornito alcun argomento, crea una sequenza vuota. Se, come argomento, viene fornito un numero intero, crea una sequenza di `bytes` contenente il numero specificato di byte di valore zero. Se, come argomento, viene fornita una sequenza, crea una sequenza di `bytes` che contiene gli elementi di tale sequenza.
  Parametro: *x*, un numero intero o una sequenza
  Restituisce: la nuova sequenza di `bytes`

- **`chr(`*x*`)`**
  Questa funzione crea una stringa contenente un unico carattere, quello il cui codice Unicode è *x*.
  Parametro: *x*, un valore di tipo intero
  Restituisce: una stringa contenente il carattere il cui codice Unicode è *x*

- **`dict()`**
- **`dict(`***`contenitore`***`)`**
  Questa funzione crea un nuovo dizionario. Se non viene fornito alcun argomento, crea un dizionario vuoto. Se *contenitore* è un dizionario, ne crea una copia, altrimenti, se *contenitore* è una sequenza di oggetti immutabili, gli elementi di tale contenitore diventano le chiavi del nuovo dizionario.
  Parametro: *contenitore*, una sequenza o un dizionario da cui viene creato il nuovo dizionario
  Restituisce: il nuovo dizionario

- **`float(`***`x`***`)`**
  Questa funzione converte una stringa o un numero intero in un valore in virgola mobile.
  Parametro: *x*, una stringa o un valore numerico
  Restituisce: un nuovo oggetto di tipo "numero in virgola mobile"

- **`hash(`***`oggetto`***`)`**
  Questa funzione crea un codice hash per l'oggetto dato. I codici hash sono usati per confrontare chiavi nei dizionari.
  Parametro: *oggetto*, un oggetto di qualunque tipo
  Restituisce: il codice hash (un numero intero)

- **`input()`**
- **`input(`***`prompt`***`)`**
  Questa funzione acquisisce una sequenza di caratteri fornita dall'utente del programma tramite la tastiera (input standard). Se viene fornito un argomento, prima di acquisire i caratteri visualizza la stringa *prompt* sulla finestra di console (output standard).
  Parametro: *prompt*, una stringa da visualizzare come messaggio per l'utente
  Restituisce: una stringa contenente i caratteri forniti dall'utente

- **`int(`***`x`***`)`**
  Questa funzione converte una stringa o un numero in un numero intero.
  Parametro: *x*, una stringa o un valore numerico
  Restituisce: un nuovo oggetto di tipo "numero intero"

- **`isinstance(`***`oggetto`***`, `***`nome`***`)`**
- **`isinstance(`***`oggetto`***`, `***`tuplaDiNomi`***`)`**
  Questa funzione determina se l'*oggetto* è un esemplare di una data classe o di una sua sottoclasse. Il secondo argomento può essere un unico nome di classe o una tupla di nomi di classi. Se viene fornita una tupla, la funzione determina se l'*oggetto* è un esemplare di una qualsiasi delle classi indicate nella tupla.
  Parametri: *oggetto*, un oggetto di qualunque tipo
  nome, un unico nome di classe
  tuplaDiNomi, una tupla di nomi di classi
  Restituisce: `True` se l'*oggetto* è un esemplare di una qualsiasi delle classi fornite come argomento, altrimenti `False`

- **`issubclass(`***`classe`***`, `***`nome`***`)`**
- **`issubclass(`***`classe`***`, `***`tuplaDiNomi`***`)`**

  Questa funzione determina se una *classe* è una sottoclasse di un'altra classe. Il secondo argomento può essere un unico nome di classe o una tupla di nomi di classi. Se viene fornita una tupla, la funzione determina se la *classe* è una sottoclasse di una qualsiasi delle classi indicate nella tupla.

  Parametri: *classe*, una classe qualsiasi
  nome, un unico nome di classe
  tuplaDiNomi, una tupla di nomi di classi

  Restituisce: `True` se la *classe* è una sottoclasse di una qualsiasi delle classi fornite come argomento, altrimenti `False`

- **`len(`***`contenitore`***`)`**

  Questa funzione restituisce il numero di elementi presenti nel *contenitore*.

  Parametro: *contenitore*, un contenitore

  Restituisce: un numero intero uguale al numero di elementi del *contenitore*

- **`list()`**
- **`list(`***`contenitore`***`)`**

  Questa funzione crea una nuova lista. Se non viene fornito alcun argomento, crea una lista vuota. Se *contenitore* è una lista, ne crea una copia. Se *contenitore* è un dizionario, crea una lista contenente le sue chiavi. Altrimenti, se *contenitore* è una sequenza, la nuova lista contiene gli elementi di tale sequenza.

  Parametro: *contenitore*, un contenitore i cui elementi vengono utilizzati per creare la nuova lista

  Restituisce: la nuova lista

- **`max(`***`argomento`*$_1$**`, `***`argomento`*$_2$**`, . . .)`**
- **`max(`***`contenitore`***`)`**

  Questa funzione restituisce il valore maggiore presente in una raccolta. Se viene fornito un solo argomento e questo è un *contenitore*, restituisce il valore maggiore presente in tale contenitore, che non può essere vuoto. Se vengono forniti più argomenti, restituisce il valore maggiore tra questi.

  Parametri: *contenitore*, un contenitore
  $argomento_1$, $argomento_2$, …, valori confrontabili di qualsiasi tipo

  Restituisce: il valore maggiore presente nella raccolta

- **`min(`***`argomento`*$_1$**`, `***`argomento`*$_2$**`, . . .)`**
- **`min(`***`contenitore`***`)`**

  Questa funzione restituisce il valore minore presente in una raccolta. Se viene fornito un solo argomento e questo è un *contenitore*, restituisce il valore minore presente in tale contenitore, che non può essere vuoto. Se vengono forniti più argomenti, restituisce il valore minore tra questi.

  Parametri: *contenitore*, un contenitore
  $argomento_1$, $argomento_2$, …, valori confrontabili di qualsiasi tipo

  Restituisce: il valore minore presente nella raccolta

- **`next(`***`iteratore`***`)`**
  Questa funzione restituisce l'elemento successivo di un contenitore o di un iteratore.
  Parametro: *iteratore*, un oggetto di tipo iteratore o comunque un contenitore che possa essere utilizzato in un ciclo `for`
  Restituisce: l'elemento successivo del contenitore o dell'iteratore; se non ci sono più elementi viene sollevata un'eccezione di tipo `StopIteration`

- **`open(`***`nomeFile`***`, `***`modalità`***`)`**
  Questa funzione apre un file di testo o binario, di nome *nomeFile*, e lo associa a un oggetto di tipo file. Un file può essere aperto in *modalità* di lettura, di scrittura o entrambe. Quando un file viene aperto in lettura, il file deve esistere, altrimenti viene sollevata un'eccezione. Quando un file viene aperto in scrittura e non esiste, viene creato un nuovo file; altrimenti, il contenuto del file esistente viene cancellato. Quando un file viene aperto in modalità "append" (per aggiungere dati alla fine di un file esistente) ma il file non esiste, ne viene creato uno nuovo; altrimenti i nuovi dati vengono aggiunti in fondo al file esistente.
  Parametri: *nomeFile*, una stringa che specifica il nome del file sul disco
  modalità, una stringa che specifica la modalità di apertura del file; le modalità consentite per un file di testo sono: lettura (`"r"`), scrittura (`"w"`), scrittura in fondo (*append*, `"a"`) e lettura/scrittura (`"r+"`); le modalità consentite per un file binario sono: lettura (`"rb"`), scrittura (`"wb"`), scrittura in fondo (*append*, `"ab"`) e lettura/scrittura (`"rb+"`)
  Restituisce: un oggetto di tipo file associato al file presente sul disco

- **`ord(`***`carattere`***`)`**
  Questa funzione restituisce il valore del codice Unicode del *carattere*.
  Parametro: *carattere*, una stringa contenente un solo carattere
  Restituisce: il valore del codice Unicode del *carattere*

- **`print()`**
- **`print(`***`argomento`*$_1$**`, `***`argomento`*$_2$**`, . . .)`**
- **`print(`***`argomento`*$_1$**`, `***`argomento`*$_2$**`, . . ., end=`***`stringa`**`, sep=`***`stringa`**`, file=`***`oggettoFile`***`)`**
  Questa funzione visualizza i suoi argomenti (*argomento*$_1$, *argomento*$_2$, …) sulla finesta di console (output standard), separati dalla *stringa* `sep` e seguiti dalla *stringa* `end`. Se non viene fornito alcun argomento, la funzione inizia semplicemente una nuova riga (cioè "va a capo"). In mancanza di valori per i parametri `end` e `sep`, gli argomenti vengono separati da uno spazio e l'ultimo argomento è seguito da un carattere *newline* (`\n`), che inizia una nuova riga. Per impedire che inizi una nuova riga, basta fornire la stringa vuota come valore del parametro `end` (cioè `end=""`). Per separare gli argomenti con una stringa diversa da uno spazio, basta fornire la stringa come valore del parametro `sep`. Per scrivere le informazioni in un file di testo anziché sull'output standard, basta fornire un oggetto di tipo file come valore del parametro `file`.
  Parametri: *argomento*$_1$, *argomento*$_2$, …, i valori da visualizzare
  `end=`*stringa*, un argomento la cui presenza indica che, dopo aver visualizzato l'ultimo argomento, verrà visualizzata la *stringa*
  `sep=`*stringa*, un argomento la cui presenza indica che *stringa* verrà visualizzata per separare ciascuna coppia di argomenti

`file=`*oggettoFile*, un argomento la cui presenza indica che le informazioni verranno scritte nel file di testo *oggettoFile*

- **`range(`***`fine`***`)`**
- **`range(`***`inizio`***`, `***`fine`***`)`**
- **`range(`***`inizio`***`, `***`fine`***`, `***`passo`***`)`**

  Questa funzione crea un contenitore di tipo sequenza di numeri interi, da usare, ad esempio, in un enunciato `for`. La sequenza di numeri inizia dal valore *inizio* e finisce al valore *fine* meno un'unità, con incrementi di valore uguale a *passo*. Se è presente il solo argomento *fine*, allora *inizio* vale 0 e *passo* vale 1. Se manca il solo argomento *passo*, allora vale 1.

  Parametri: *inizio*, un numero intero che indica il primo valore della sequenza
  fine, un numero intero che, diminuito di un'unità, indica l'ultimo valore della sequenza
  passo, un numero intero che indica la differenza tra due valori consecutivi della sequenza

  Restituisce: un oggetto di tipo iteratore, da poter usare in un enunciato `for`

- **`round(`***`valore`***`)`**
- **`round(`***`valore`***`, `***`cifre`***`)`**

  Questa funzione arrotonda un *valore* numerico in modo che abbia un dato numero di *cifre* decimali. Se viene fornito il solo *valore*, questo viene arrotondato all'intero più vicino.

  Parametri: *valore*, il numero (intero o in virgola mobile) che deve essere arrotondato
  cifre, il numero di cifre decimali

  Restituisce: il *valore* arrotondato all'intero più vicino o al numero dato di *cifre* decimali

- **`set()`**
- **`set(`***`contenitore`***`)`**

  Questa funzione crea un nuovo insieme. Se non viene fornito alcun argomento, crea un insieme vuoto. Se *contenitore* è un insieme, ne crea una copia. Se *contenitore* è un dizionario, crea un insieme contenente le sue chiavi. Altrimenti, se *contenitore* è una sequenza, il nuovo insieme contiene gli elementi di tale sequenza.

  Parametro: *contenitore*, un contenitore i cui elementi vengono utilizzati per creare il nuovo insieme

  Restituisce: il nuovo insieme

- **`sorted(`***`contenitore`***`)`**

  Questa funzione crea una lista ordinata in senso crescente contenente gli elementi del *contenitore*.

  Parametro: *contenitore*, un contenitore i cui elementi vengono utilizzati per creare la nuova lista ordinata

  Restituisce: la nuova lista ordinata

- **`str(`***`oggetto`***`)`**
  Questa funzione converte un *oggetto* in una stringa.
  Parametro: *oggetto*, l'oggetto da convertire in una stringa
  Restituisce: la nuova stringa

- **`sum(`***`contenitore`***`)`**
  Questa funzione calcola la somma degli elementi del *contenitore*, che deve contenere numeri.
  Parametro: *contenitore*, un contenitore di valori numerici da sommare
  Restituisce: la somma degli elementi del contenitore

- **`super()`**
  Quando un metodo viene invocato con l'oggetto restituito da questa funzione, viene in realtà invocato l'omonimo metodo della superclasse.

- **`tuple()`**
- **`tuple(`***`contenitore`***`)`**
  Questa funzione crea una nuova tupla. Se non viene fornito alcun argomento, crea una tupla vuota. Se *contenitore* è una tupla, ne crea una copia. Se *contenitore* è un dizionario, crea una tupla contenente le sue chiavi. Altrimenti, se *contenitore* è una sequenza, la nuova tupla contiene gli elementi di tale sequenza.
  Parametro: *contenitore*, un contenitore i cui elementi vengono utilizzati per creare la nuova tupla
  Restituisce: la nuova tupla

Classi predefinite

Classe `dict`

- *`d`* **`= dict()`**
- *`d`* **`= dict(`***`c`***`)`**
  Questa funzione crea un nuovo dizionario. Se non viene fornito alcun argomento, crea un dizionario vuoto. Se *c* è un dizionario, ne crea una copia, altrimenti, se *c* è una sequenza di oggetti immutabili, gli elementi di tale contenitore diventano le chiavi del nuovo dizionario.
  Parametro: *c*, una sequenza o un dizionario da cui viene creato il nuovo dizionario
  Restituisce: il nuovo dizionario

- *`valore`* **`=`** *`d`***`[`***`chiave`***`]`**
  L'operatore `[]` restituisce il valore associato, nel dizionario *d*, alla *chiave*, che deve esistere, altrimenti viene sollevata un'eccezione.

- *`d`***`[`***`chiave`***`]`** **`=`** *`valore`*
  L'operatore `[]` aggiunge al dizionario *d* una nuova coppia *chiave*/*valore*, se *chiave* non è presente nel dizionario. Se, invece, *chiave* è già presente nel dizionario, questa viene associata al nuovo *valore*.

- *chiave* **in** *d*
- *chiave* **not in** *d*
  Gli operatori `in`/`not in` determinano, rispettivamente, se la *chiave* appartiene o non appartiene al dizionario *d*.

- **len(***d***)**
  Questa funzione restituisce il numero di coppie presenti nel dizionario *d*.
  Parametro: *d*, un dizionario
  Restituisce: un numero intero uguale al numero di coppie presenti nel dizionario

- *d***.clear()**
  Questo metodo elimina tutte le coppie dal dizionario *d*.

- *d***.get(***chiave***,** *predefinito***)**
  Questo metodo restituisce il valore associato alla *chiave* nel dizionario *d*; se la *chiave* non è presente nel dizionario *d*, restituisce il valore *predefinito*.
  Parametri: *chiave*, la chiave da cercare
  predefinito, il valore restituito se la *chiave* non è presente nel dizionario *d*
  Restituisce: il valore associato alla chiave oppure il valore predefinito se la chiave non è presente nel dizionario

- *d***.items()**
  Questo metodo restituisce una lista contenente tutte le coppie presenti nel dizionario *d*. La lista contiene una tupla per ogni coppia: il primo elemento di ciascuna tupla è una delle chiavi, mentre il secondo elemento è il valore associato a quella chiave.
  Restituisce: una lista di tuple contenenti le coppie del dizionario

- *d***.keys()**
  Questo metodo restituisce una lista contenente tutte le chiavi presenti nel dizionario *d*.
  Restituisce: una lista contenente le chiavi del dizionario

- *d***.pop(***chiave***)**
  Questo metodo elimina dal dizionario *d* la *chiave* e il valore associato a questa.
  Parametro: *chiave*, la chiave da eliminare
  Restituisce: il valore associato alla chiave

- *d***.values()**
  Questo metodo restituisce una lista contenente tutti i valori presenti nel dizionario *d*.
  Restituisce: una lista contenente i valori del dizionario

Classe `list`

- *l* **= list()**
- *l* **= list(***sequenza***)**
  Questa funzione crea una nuova lista vuota oppure contenente tutti gli elementi della *sequenza*.

Parametro: *sequenza*, una sequenza da cui viene creata la nuova lista
Restituisce: la nuova lista

- `valore = l[posizione]`
  L'operatore `[]` restituisce l'elemento che, nella lista *l*, si trova in corrispondenza dell'indice *posizione*, che deve appartenere all'intervallo degli indici validi, altrimenti viene sollevata un'eccezione.

- `l[posizione] = valore`
  L'operatore `[]` sostituisce con il *valore* l'elemento che, nella lista *l*, si trova in corrispondenza dell'indice *posizione*, che deve appartenere all'intervallo degli indici validi, altrimenti viene sollevata un'eccezione.

- `elemento in l`
- `elemento not in l`
  Gli operatori `in`/`not in` determinano, rispettivamente, se l'*elemento* appartiene o non appartiene alla lista *l*.

- `len(l)`
  Questa funzione restituisce il numero di elementi presenti nella lista *l*.
  Parametro: *l*, una lista
  Restituisce: un numero intero uguale al numero di elementi presenti nella lista

- `l.append(elemento)`
  Questo metodo aggiunge *elemento* in fondo alla lista *l*.
  Parametro: *elemento*, l'elemento da aggiungere

- `l.index(elemento)`
  Questo metodo restituisce la posizione dell'*elemento* nella lista *l*. L'elemento deve essere presente nella lista, altrimenti viene sollevata un'eccezione.
  Parametro: *elemento*, l'elemento da localizzare
  Restituisce: la posizione dell'elemento all'interno della lista

- `l.insert(posizione, elemento)`
  Questo metodo inserisce l'*elemento* in corrispondenza dell'indice *posizione* nella lista *l*. Tutti gli elementi della lista che si trovano in corrispondenza di indici non minori di *posizione* vengono spostati nella posizione di indice immediatamente superiore.
  Parametri: *posizione*, la posizione in cui va inserito l'elemento
  elemento, l'elemento da inserire

- `l.pop()`
- `l.pop(posizione)`
  Questo metodo elimina dalla lista *l* l'ultimo elemento oppure l'elemento che si trova in corrispondenza dell'indice *posizione*. Tutti gli elementi della lista che si trovano in corrispondenza di indici maggiori di *posizione* vengono spostati nella posizione di indice immediatamente inferiore.
  Parametro: *posizione*, la posizione dell'elemento da eliminare
  Restituisce: l'elemento eliminato

- **`l.remove(elemento)`**
  Questo metodo elimina *elemento* dalla lista *l*. L'*elemento* deve essere presente nella lista, altrimenti viene sollevata un'eccezione. Tutti gli elementi della lista che si trovano in corrispondenza di indici maggiori di quello dell'*elemento* vengono spostati nella posizione di indice immediatamente inferiore.
  Parametro: *elemento*, l'elemento da eliminare

- **`l.sort()`**
  Questo metodo ordina in senso crescente gli elementi della lista *l*.

Classe `set`

- **`s = set()`**
- **`s = set(sequenza)`**
  Questa funzione crea un nuovo insieme vuoto o contenente gli elementi presenti nella *sequenza*. Se la *sequenza* contiene valori duplicati, viene aggiunto all'insieme un solo esemplare di ciascun valore.
  Parametro: *sequenza*, una sequenza da cui viene creato il nuovo insieme
  Restituisce: il nuovo insieme

- **`elemento in s`**
- **`elemento not in s`**
  Gli operatori `in`/`not in` determinano, rispettivamente, se l'*elemento* appartiene o non appartiene all'insieme *s*.

- **`len(s)`**
  Questa funzione restituisce il numero di elementi presenti nell'insieme *s*.
  Parametro: *s*, un insieme
  Restituisce: un numero intero uguale al numero di elementi presenti nell'insieme

- **`s.add(elemento)`**
  Questo metodo aggiunge l'*elemento* all'insieme *s*. Se l'*elemento* appartiene già all'insieme, non viene intrapresa alcuna azione.
  Parametro: *elemento*, il nuovo elemento

- **`s.clear()`**
  Questo metodo elimina tutti gli elementi dall'insieme *s*.

- **`s.difference(t)`**
  Questo metodo crea un nuovo insieme che contiene gli elementi dell'insieme *s* che non appartengono all'insieme *t*.
  Parametro: *t*, un insieme
  Restituisce: il nuovo insieme risultante dalla differenza tra insiemi

- **`s.discard(elemento)`**
  Questo metodo elimina l'*elemento* dall'insieme *s*. Se l'*elemento* non appartiene all'insieme *s*, non viene intrapresa alcuna azione.
  Parametro: *elemento*, l'elemento che deve essere eliminato dall'insieme

- **`s.intersection(t)`**
  Questo metodo crea un nuovo insieme che contiene gli elementi che appartengono sia all'insieme *s* sia all'insieme *t*.
  Parametro: *t*, un insieme
  Restituisce: il nuovo insieme risultante dall'intersezione tra insiemi

- **`s.issubset(t)`**
  Questo metodo determina se l'insieme *s* è un sottoinsieme dell'insieme *t*.
  Parametro: *t*, un insieme
  Restituisce: `True` se *s* è un sottoinsieme di *t*, altrimenti `False`

- **`s.remove(elemento)`**
  Questo metodo elimina l'*elemento* dall'insieme *s*. Se l'*elemento* non appartiene all'insieme *s*, viene sollevata un'eccezione.
  Parametro: *elemento*, l'elemento che deve essere eliminato dall'insieme

- **`s.union(t)`**
  Questo metodo crea un nuovo insieme che contiene gli elementi che appartengono all'insieme *s* o all'insieme *t*, o a entrambi.
  Parametro: *t*, un insieme
  Restituisce: il nuovo insieme risultante dall'unione tra insiemi

Classe `str`

- **`s = str()`**
- **`s = str(oggetto)`**
  Questa funzione crea una nuova stringa vuota oppure risultante dalla conversione in stringa dell'*oggetto*.
  Parametro: *oggetto*, l'oggetto da convertire in stringa
  Restituisce: la nuova stringa

- *`sottostringa`* **`in`** *`s`*
- *`sottostringa`* **`not in`** *`s`*
  Gli operatori `in`/`not in` determinano, rispettivamente, se la *sottostringa* è presente o non è presente nella stringa *s*.

- **`len(s)`**
  Questa funzione restituisce la lunghezza della stringa *s*.
  Parametro: *s*, una stringa
  Restituisce: un numero intero uguale al numero di caratteri presenti nella stringa

- **`s.count(sottostringa)`**
  Questo metodo restituisce il numero di occorrenze non sovrapposte della *sottostringa* nella stringa *s*.
  Parametro: *sottostringa*, la sottostringa da cercare
  Restituisce: il numero di occorrenze non sovrapposte della sottostringa nella stringa

- **`s.endswith(sottostringa)`**
  Questo metodo determina se la stringa *s* termina con la *sottostringa*.
  Parametro: *sottostringa*, la sottostringa da cercare
  Restituisce: `True` se la stringa termina con la sottostringa, altrimenti `False`

- **`s.find(sottostringa)`**
- **`s.find(sottostringa, inizio)`**
  Questo metodo restituisce il minimo valore dell'indice in cui inizia la *sottostringa* all'interno della stringa *s*, oppure `-1` se la *sottostringa* non è presente in *s*. Se è presente il parametro *inizio*, la ricerca nella stringa *s* viene effettuata a partire dalla posizione *inizio*, fino alla fine della stringa.
  Parametri: *sottostringa*, la sottostringa da cercare
  inizio, la posizione in *s* da cui inizia la ricerca
  Restituisce: la posizione in cui la sottostringa inizia nella stringa

- **`s.isalnum()`**
  Questo metodo verifica se la stringa *s* è costituita soltanto da lettere e cifre, e contiene almeno un carattere.
  Restituisce: `True` se entrambe le condizioni sono vere, altrimenti `False`

- **`s.isalpha()`**
  Questo metodo verifica se la stringa *s* è costituita soltanto da lettere e contiene almeno un carattere.
  Restituisce: `True` se entrambe le condizioni sono vere, altrimenti `False`

- **`s.isdigit()`**
  Questo metodo verifica se la stringa *s* è costituita soltanto da cifre e contiene almeno un carattere.
  Restituisce: `True` se entrambe le condizioni sono vere, altrimenti `False`

- **`s.islower()`**
  Questo metodo verifica se la stringa *s* è costituita soltanto da lettere minuscole e contiene almeno un carattere.
  Restituisce: `True` se entrambe le condizioni sono vere, altrimenti `False`

- **`s.isspace()`**
  Questo metodo verifica se la stringa *s* è costituita soltanto da caratteri di spaziatura (spazi, caratteri di tabulazione e caratteri di "nuova riga") e contiene almeno un carattere.
  Restituisce: `True` se entrambe le condizioni sono vere, altrimenti `False`

- **`s.isupper()`**
  Questo metodo verifica se la stringa *s* è costituita soltanto da lettere maiuscole e contiene almeno un carattere.
  Restituisce: `True` se entrambe le condizioni sono vere, altrimenti `False`

- **`s.lower()`**
  Questo metodo restituisce una nuova stringa che è la versione minuscola della stringa *s*.
  Restituisce: una nuova stringa che è la versione minuscola della stringa

- **`s.lstrip()`**
- **`s.lstrip(`*`caratteri`*`)`**

  Questo metodo restituisce una nuova versione della stringa *s* nella quale sono stati eliminati tutti i caratteri di spaziatura (spazi, caratteri di tabulazione e caratteri di "nuova riga") iniziali, cioè all'estremità "sinistra" ("left", da cui la lettera l). Se viene fornito un argomento, invece dei caratteri di spaziatura vengono eliminati i caratteri presenti nella stringa *caratteri*.

  Parametro: *caratteri*, una stringa che specifica i caratteri da eliminare
  Restituisce: una nuova versione della stringa

- **`s.replace(`*`vecchia`*`, `*`nuova`*`)`**

  Questo metodo restituisce una nuova versione della stringa *s* nella quale tutte le occorrenze della stringa *vecchia* sono state sostituite dalla stringa *nuova*.

  Parametri: *vecchia*, la sottostringa da sostituire
  nuova, la sottostringa che va a sostituire la vecchia
  Restituisce: una nuova versione della stringa

- **`s.rstrip()`**
- **`s.rstrip(`*`caratteri`*`)`**

  Questo metodo restituisce una nuova versione della stringa *s* nella quale sono stati eliminati tutti i caratteri di spaziatura (spazi, caratteri di tabulazione e caratteri di "nuova riga") finali, cioè all'estremità "destra" ("right", da cui la lettera r). Se viene fornito un argomento, invece dei caratteri di spaziatura vengono eliminati i caratteri presenti nella stringa *caratteri*.

  Parametro: *caratteri*, una stringa che specifica i caratteri da eliminare
  Restituisce: una nuova versione della stringa

- **`s.rsplit(`*`sep`*`, `*`maxSuddivisioni`*`)`**

  Questo metodo restituisce una lista di sottostringhe generate suddividendo la stringa *s* a partire dalla fine (cioè dall'estremità "destra","right", da cui la lettera r) e usando la sottostringa *sep* come separatore. Al massimo vengono generate *maxSuddivisioni* + 1 sottostringhe.

  Parametri: *sep*, la sottostringa che specifica il separatore usato per la suddivisione
  maxSuddivisioni, il massimo numero di suddivisioni
  Restituisce: una lista di sottostringhe generate suddividendo la stringa

- **`s.split()`**
- **`s.split(`*`sep`*`)`**
- **`s.split(`*`sep`*`, `*`maxSuddivisioni`*`)`**

  Questo metodo restituisce una lista di sottostringhe generate suddividendo la stringa *s* a partire dall'inizio e usando la sottostringa *sep* come separatore (se il parametro *sep* non è presente, viene usata una stringa contenente soltanto uno spazio). Se è presente il parametro *maxSuddivisioni*, al massimo vengono generate *maxSuddivisioni* + 1 sottostringhe.

  Parametri: *sep*, la sottostringa che specifica il separatore usato per la suddivisione
  maxSuddivisioni, il massimo numero di suddivisioni
  Restituisce: una lista di sottostringhe generate suddividendo la stringa

- **`s.splitlines()`**
  Questo metodo restituisce una lista contenente le singole righe ottenute suddividendo la stringa *s* usando il carattere di "nuova riga" (`\n`) come elemento separatore.
  Restituisce: una lista di righe generate suddividendo la stringa

- **`s.startswith(sottostringa)`**
  Questo metodo determina se la stringa *s* inizia con la *sottostringa*.
  Parametro: *sottostringa*, la sottostringa da cercare
  Restituisce: `True` se la stringa inizia con la sottostringa, altrimenti `False`

- **`s.strip()`**
- **`s.strip(caratteri)`**
  Questo metodo restituisce una nuova versione della stringa *s* nella quale sono stati eliminati tutti i caratteri di spaziatura (spazi, caratteri di tabulazione e caratteri di "nuova riga") iniziali e finali. Se viene fornito un argomento, invece dei caratteri di spaziatura vengono eliminati i caratteri presenti nella stringa *caratteri*.
  Parametro: *caratteri*, una stringa che specifica i caratteri da eliminare
  Restituisce: una nuova versione della stringa

- **`s.upper()`**
  Questo metodo restituisce una nuova stringa che è la versione maiuscola della stringa *s*.
  Restituisce: una nuova stringa che è la versione maiuscola della stringa

Gestione di file (input/output)

Metodi e funzioni comuni

Questi metodi e queste funzioni sono comuni ai file di testo e ai file binari.

- **`open(nomeFile, modalità)`**
  Questa funzione apre un file di testo o binario, di nome *nomeFile*, e lo associa a un oggetto di tipo file. Un file può essere aperto in *modalità* di lettura, di scrittura o entrambe. Quando un file viene aperto in lettura, il file deve esistere, altrimenti viene sollevata un'eccezione. Quando un file viene aperto in scrittura e non esiste, viene creato un nuovo file; altrimenti, il contenuto del file esistente viene cancellato. Quando un file viene aperto in modalità "append" (per aggiungere dati alla fine di un file esistente) ma il file non esiste, ne viene creato uno nuovo; altrimenti i nuovi dati vengono aggiunti in fondo al file esistente.
  Parametri: *nomeFile*, una stringa che specifica il nome del file sul disco
  modalità, una stringa che specifica la modalità di apertura del file; le modalità consentite per un file di testo sono: lettura (`"r"`), scrittura (`"w"`), scrittura in fondo (*append*, `"a"`) e lettura/scrittura (`"r+"`); le modalità consentite per un file binario sono: lettura (`"rb"`), scrittura (`"wb"`), scrittura in fondo (*append*, `"ab"`) e lettura/scrittura (`"rb+"`)
  Restituisce: un oggetto di tipo file associato al file presente sul disco

- **`f.close()`**
  Questo metodo chiude il file rappresentato dall'oggetto *f*, se è aperto; non ha alcun effetto se il file è già stato chiuso.

- **`f.seek(`*`scostamento`*`, `*`rispettoA`*`)`**
  Questo metodo sposta il cursore all'interno del file *f* di un numero di byte uguale a *scostamento*. L'argomento *rispettoA* indica se lo spostamento avviene rispetto all'inizio del file (`SEEK_SET`), all'attuale posizione del cursore (`SEEK_CUR`) o alla fine del file (`SEEK_END`).
  Parametri: *scostamento*, un numero intero che indica di quanti byte spostare il cursore
  rispettoA, una delle costanti `SEEK_SET`, `SEEK_CUR` o `SEEK_END` (definite nel modulo `os`)
  Restituisce: la nuova posizione assoluta del cursore, relativa all'inizio del file

- **`f.tell()`**
  Questo metodo restituisce l'attuale posizione del cursore nel file *f*.
  Restituisce: la posizione assoluta del cursore, relativa all'inizio del file

Metodi per i file di testo

Questi metodi possono essere usati soltanto con i file di testo.

- **`f.read()`**
- **`f.read(`*`numero`*`)`**
  Questo metodo legge dal file di testo *f* aperto in lettura i successivi caratteri e li restituisce sotto forma di stringa. Se non viene fornito alcun argomento, viene letto l'intero file e il suo contenuto viene restituito come un'unica stringa, altrimenti viene letta una quantità di caratteri uguale a *numero*. Se dal file sono già stati letti tutti i caratteri disponibili, viene restituita una stringa vuota.
  Parametro: *numero*, un numero intero che indica quanti caratteri devono essere letti
  Restituisce: una stringa contenente i caratteri letti dal file

- **`f.readline()`**
  Questo metodo legge dal file di testo *f* aperto in lettura la successiva riga di testo e la restituisce sotto forma di stringa. Se è già stata raggiunta la fine del file, viene restituita una stringa vuota.
  Restituisce: una stringa contenente i caratteri letti dal file

- **`f.readlines()`**
  Questo metodo legge dal file di testo *f* aperto in lettura la porzione di testo rimanente e la restituisce sotto forma di una lista di stringhe, ciascuna delle quali contiene una riga del testo letto dal file.
  Restituisce: una lista di stringhe contenenti le righe di testo lette dal file

- **`f.write(`*`stringa`*`)`**
  Questo metodo scrive la *stringa* nel file di testo *f* aperto in scrittura.
  Parametro: *stringa*, la stringa da scrivere

Metodi per i file binari

Questi metodi possono essere usati soltanto con i file binari.

- **`f.read()`**
- **`f.read(`***`numero`***`)`**
  Questo metodo legge dal file binario *f* aperto in lettura i successivi byte e li restituisce sotto forma di sequenza di `bytes`. Se non viene fornito alcun argomento, viene letto l'intero file e il suo contenuto viene restituito come un'unica sequenza di `bytes`, altrimenti viene letta una quantità di byte uguale a *numero*.
  Parametro: *numero*, un numero intero che indica quanti byte devono essere letti
  Restituisce: una sequenza di `bytes` contenente i byte letti dal file

- **`f.write(`***`dati`***`)`**
  Questo metodo scrive nel file binario *f*, aperto in scrittura, la sequenza di `bytes` *dati*.
  Parametro: *dati*, una sequenza di `bytes`

Il modulo `csv`

Classe `reader`

- ***`r`*** **`= reader(`*****`nomeFile`*****`)`**
  Questa funzione crea un nuovo oggetto di tipo "lettore", da utilizzare per scandire il contenuto di un file CSV. Se il file non esiste viene sollevata un'eccezione.
  Parametro: *nomeFile*, una stringa contenente il nome del file CSV
  Restituisce: un oggetto di tipo "lettore di CSV"

Classe `writer`

- ***`w`*** **`= writer(`*****`nomeFile`*****`)`**
  Questa funzione crea un nuovo oggetto di tipo "scrittore", da utilizzare per creare un nuovo file CSV. Se il file non esiste viene creato un nuovo file, altrimenti il contenuto del file esistente viene cancellato.
  Parametro: *nomeFile*, una stringa contenente il nome del file CSV
  Restituisce: un oggetto di tipo "scrittore di CSV"

- ***`w`*****`.writerow(`*****`riga`*****`)`**
  Questo metodo aggiunge una riga di dati al file CSV rappresentato dall'oggetto *w*. I dati contenuti in *riga* devono essere una sequenza di stringhe o numeri.
  Parametro: *riga*, la sequenza di stringhe o numeri che costituiscono la riga da scrivere

Il modulo `email.mime.multipart`

- ***`m`*** **`= MIMEMultipart()`**
  Questa funzione crea e restituisce un nuovo oggetto di tipo `Message` di MIME, usato per creare messaggi di posta elettronica (*e-mail*).
  Restituisce: un oggetto di tipo `Message`

Classe `Message`

- **`m.add_header(`***`campo, valore`***`)`**
  Questo metodo aggiunge al messaggio *m* una nuova riga di intestazione, costituita dal *campo* e dal relativo *valore*.
  Parametri: *campo*, una stringa contenente il nome del campo d'intestazione da aggiungere
  valore, una stringa contenente il valore associato al *campo*

- **`m.attach(`***`contenuto`***`)`**
  Questo metodo aggiunge *contenuto* (che deve essere un oggetto di tipo "contenuto MIME") al corpo del messaggio *m*.
  Parametro: *contenuto*, un oggetto di tipo "contenuto MIME" contenente le informazioni da aggiungere al corpo del messaggio

Il modulo `email.mime.text`

- **`MIMEText(`***`dati, tipo`***`)`**
  Questa funzione crea un nuovo oggetto di tipo "contenuto MIME", contenente il testo *dati*.
  Parametri: *dati*, una stringa contenente il testo da inserire nell'oggetto
  tipo, una stringa contenente la codifica MIME del tipo di testo (ad esempio, `"plain"` o `"text"`)
  Restituisce: un oggetto di tipo "contenuto MIME"

Il modulo `email.mime.image`

- **`MIMEImage(`***`dati`***`)`**
  Questa funzione crea un nuovo oggetto di tipo "contenuto MIME", contenente i *dati* relativi a un'immagine.
  Parametro: *dati*, una stringa contenente i dati relativi a un'immagine
  Restituisce: un oggetto di tipo "contenuto MIME"

Il modulo `email.mime.application`

- **`MIMEApplication(`***`dati`***`)`**
  Questa funzione crea un nuovo oggetto di tipo "contenuto MIME", contenente i *dati* relativi a un tipo di dato specifico per un'applicazione, ad esempio un documento PDF o un foglio elettronico.
  Parametro: *dati*, una stringa contenente i dati da memorizzare
  Restituisce: un oggetto di tipo "contenuto MIME"

Il modulo `json`

- **`dumps(d)`**
  Questa funzione converte in una stringa avente il formato JSON l'intero contenuto del dizionario *d*.
  Parametro: *d*, un dizionario da cui viene creata una stringa in formato JSON
  Restituisce: una stringa in formato JSON

- **`loads(s)`**
  Questa funzione converte una stringa avente il formato JSON in un dizionario. I valori numerici vengono convertiti nel tipo di dato più appropriato (numero intero o in virgola mobile).
  Parametro: *s*, una stringa in formato JSON
  Restituisce: un dizionario contenente i dati estratti dalla stringa in formato JSON

Il modulo `math`

- **`e`**
  Questa costante è il valore di *e*, la base dei logaritmi naturali.

- **`pi`**
  Questa costante è il valore di π.

- **`acos(x)`**
  Questa funzione restituisce l'angolo avente *x* come coseno, cioè $\cos^{-1}x \in [0, \pi]$.
  Parametro: *x*, un valore in virgola mobile compreso tra –1 e 1
  Restituisce: l'arcocoseno dell'argomento, in radianti

- **`asin(x)`**
  Questa funzione restituisce l'angolo avente *x* come seno, cioè $\sin^{-1}x \in [-\pi/2, \pi/2]$.
  Parametro: *x*, un valore in virgola mobile compreso tra –1 e 1
  Restituisce: l'arcoseno dell'argomento, in radianti

- **`atan(x)`**
  Questa funzione restituisce l'angolo avente *x* come tangente, cioè $\tan^{-1}x \in (-\pi/2, \pi/2)$.
  Parametro: *x*, un valore in virgola mobile
  Restituisce: l'arcotangente dell'argomento, in radianti

- **`atan2(y, x)`**
  Questa funzione restituisce l'angolo avente y/x come tangente, cioè $\tan^{-1}(y/x) \in (-\pi, \pi)$. Se *x* può essere uguale a zero oppure se è necessario distinguere l'orientamento "nord ovest" da quello "sud est", allora si usa questa funzione invece della funzione `atan(y/x)`.
  Parametri: *y*, *x*, due valori in virgola mobile
  Restituisce: l'angolo, in radianti, tra l'asse *x* del piano cartesiano e la retta che passa per i punti (0, 0) e (*x*, *y*)

- **`ceil(x)`**
  Questa funzione restituisce (come numero in virgola mobile) il minimo numero intero $\geq x$.
  Parametro: x, un valore in virgola mobile
  Restituisce: il minimo numero intero $\geq x$

- **`cos(x)`**
  Questa funzione restituisce il coseno dell'angolo x, espresso in radianti.
  Parametro: x, un angolo espresso in radianti
  Restituisce: il coseno dell'argomento

- **`degrees(x)`**
  Questa funzione converte in gradi un valore espresso in radianti.
  Parametro: x, un angolo espresso in radianti
  Restituisce: l'angolo espresso in gradi

- **`exp(x)`**
  Questa funzione restituisce il valore e^x, dove e è la base dei logaritmi naturali.
  Parametro: x, un valore in virgola mobile
  Restituisce: e^x

- **`fabs(x)`**
  Questa funzione restituisce, come numero in virgola mobile, il valore assoluto di x, cioè $|x|$.
  Parametro: x, un valore numerico
  Restituisce: il valore assoluto dell'argomento, come numero in virgola mobile

- **`factorial(x)`**
  Questa funzione restituisce $x!$, cioè il fattoriale di x.
  Parametro: x, un numero intero non negativo
  Restituisce: il fattoriale dell'argomento

- **`floor(x)`**
  Questa funzione restituisce (come numero in virgola mobile) il massimo numero intero $\leq x$.
  Parametro: x, un valore in virgola mobile
  Restituisce: il massimo numero intero $\leq x$

- **`hypot(x, y)`**
  Questa funzione restituisce la norma euclidea $\sqrt{x^2 + y^2}$
  Parametri: x, y, due valori numerici
  Restituisce: la lunghezza del segmento che congiunge l'origine delle coordinate cartesiane e il punto (x, y)

- **`log(x)`**
- **`log(x, base)`**
  Questa funzione restituisce il logaritmo naturale (cioè in base e) di x oppure, se viene fornito il secondo argomento, il logaritmo di x in base *base*.

Parametri: *x*, un numero positivo
base, un numero positivo diverso da 1
Restituisce: il logaritmo dell'argomento

- **`log2(x)`**
  Questa funzione restituisce il logaritmo di *x* in base 2.
  Parametro: *x*, un numero positivo
  Restituisce: il logaritmo in base 2 dell'argomento

- **`log10(x)`**
  Questa funzione restituisce il logaritmo di *x* in base 10.
  Parametro: *x*, un numero positivo
  Restituisce: il logaritmo in base 10 dell'argomento

- **`radians(x)`**
  Questa funzione converte in radianti un valore espresso in gradi.
  Parametro: *x*, un angolo espresso in gradi
  Restituisce: l'angolo espresso in radianti

- **`sin(x)`**
  Questa funzione restituisce il seno dell'angolo *x*, espresso in radianti.
  Parametro: *x*, un angolo espresso in radianti
  Restituisce: il seno dell'argomento

- **`sqrt(x)`**
  Questa funzione restituisce la radice quadrata di *x*.
  Parametro: *x*, un numero in virgola mobile non negativo
  Restituisce: la radice quadrata dell'argomento

- **`tan(x)`**
  Questa funzione restituisce la tangente dell'angolo *x*, espresso in radianti.
  Parametro: *x*, un angolo espresso in radianti
  Restituisce: la tangente dell'argomento

- **`trunc(x)`**
  Questa funzione restituisce la parte intera di *x*.
  Parametri: *x*, un numero
  Restituisce: la parte intera dell'argomento

Il modulo os

- **`SEEK_CUR`**
- **`SEEK_END`**
- **`SEEK_SET`**
  Queste costanti sono usate nell'invocazione del metodo `seek` per specificare la posizione relativamente alla quale va spostato il cursore di un file: `SEEK_CUR` indica che lo spostamento è relativo alla posizione attuale del cursore, `SEEK_END` indica che è relativo alla fine del file e `SEEK_SET` che è relativo all'inizio.

- **`chdir(`***`percorso`***`)`**
  Questa funzione modifica la cartella di lavoro attuale, che diventa *percorso*.
  Parametro: *percorso*, una stringa contenente il nome (assoluto o relativo) di una cartella

- **`getcwd()`**
  Questa funzione restituisce il nome completo del percorso rappresentato dalla cartella di lavoro attuale.
  Restituisce: una stringa contenente il nome della cartella di lavoro attuale

- **`listdir()`**
- **`listdir(`***`percorso`***`)`**
  Questa funzione restituisce una lista contenente i nomi degli elementi (file o sotto-cartelle) presenti nella cartella indicata dal *percorso* o dalla cartella di lavoro attuale (se manca il *percorso*).
  Parametro: *percorso*, una stringa contenente il nome (assoluto o relativo) di una cartella
  Restituisce: una lista di stringhe contenente i nomi degli elementi presenti nella cartella indicata

- **`remove(`***`nomeFile`***`)`**
  Questa funzione cancella un file esistente.
  Parametro: *nomeFile*, una stringa contenente il nome (assoluto o relativo) di un file esistente

- **`rename(`***`vecchio, nuovo`***`)`**
  Questa funzione assegna il *nuovo* nome al *vecchio* file.
  Parametri: *vecchio*, una stringa contenente il nome (assoluto o relativo) di un file esistente
  nuovo, una stringa contenente il nuovo nome (assoluto o relativo) del file

Il modulo `os.path`

- **`exists(`***`percorso`***`)`**
  Questa funzione determina se il *percorso* fa riferimento a un file o a una cartella esistente.
  Parametro: *percorso*, una stringa contenente il nome (assoluto o relativo) di una cartella o di un file
  Restituisce: `True` se il file o la cartella esiste, altrimenti `False`

- **`getsize(`***`percorso`***`)`**
  Questa funzione restituisce la dimensione del file il cui nome è indicato dal *percorso*.
  Parametro: *percorso*, una stringa contenente il nome (assoluto o relativo) di un file
  Restituisce: la dimensione del file, in byte

- **`isdir(`***`percorso`***`)`**
  Questa funzione determina se *percorso* è il nome di una cartella.
  Parametro: *percorso*, una stringa contenente il nome (assoluto o relativo) di una cartella
  Restituisce: `True` se *percorso* è il nome di una cartella, altrimenti `False`

- **`isfile(`***`percorso`***`)`**
  Questa funzione determina se *percorso* è il nome di un file.
  Parametro: *percorso*, una stringa contenente il nome (assoluto o relativo) di un file
  Restituisce: `True` se *percorso* è il nome di un file, altrimenti `False`

- **`join(`***`percorso, nome`***`)`**
  Questa funzione aggiunge al *percorso* il *nome* di un file o di una cartella, inserendo tra le due stringhe il separatore di percorso appropriato per il sistema operativo in uso.
  Parametri: *percorso*, una stringa contenente il nome (assoluto o relativo) di una cartella
  nome, una stringa contenente il nome del file o della cartella da aggiungere
  Restituisce: una stringa contenente il nome del nuovo percorso

Il modulo `random`

- **`randint(`***`primo, ultimo`***`)`**
  Questa funzione restituisce il successivo numero intero di una sequenza pseudo-casuale uniformemente distribuita nell'intervallo [*primo*, *ultimo*].
  Parametri: *primo*, *ultimo*, il primo e l'ultimo valore dell'intervallo di numeri interi
  Restituisce: il successivo numero intero pseudo-casuale ≥ *primo* e ≤ *ultimo*

- **`random()`**
  Questa funzione restituisce il successivo numero in virgola mobile di una sequenza pseudo-casuale uniformemente distribuita nell'intervallo [0.0, 1.0).
  Restituisce: il successivo numero pseudo-casuale in virgola mobile ≥ 0.0 e < 1.0

Il modulo `re`

- **`split(`***`schema, stringa`***`)`**
  Questa funzione suddivide la *stringa* in corrispondenza di ciascuna occorrenza dell'espressione canonica *schema* e restituisce una lista con le sottostringhe generate dalla suddivisione.
  Parametri: *schema*, una stringa contenente un'espressione canonica usata per determinare i punti in cui suddividere la stringa data
  stringa, una stringa da suddividere sulla base dello schema fornito
  Restituisce: una lista contenente le sottostringhe generate dalla suddivisione

Il modulo `shutil`

- **`copy(`***`sorgente, destinazione`***`)`**
  Questa funzione copia l'intero contenuto del file *sorgente* nel nuovo file *destinazione* o in un nuovo file di nome *sorgente* nella cartella *destinazione*.
  Parametri: *sorgente*, una stringa contenente il nome (assoluto o relativo) di un file esistente
  destinazione, una stringa contenente il nome (assoluto o relativo) di un file o di una cartella esistente

Il modulo `smtplib`

Classe `SMTP`

- ***`c`*** **`= SMTP(`***`host, porta`***`)`**
  Questa funzione crea un nuovo oggetto di tipo "connessione SMTP" per instaurare una connessione con un server di posta elettronica, per inviare un messaggio.
  Parametri: *host*, una stringa contenente il nome del computer che ospita il server
  porta, un numero intero che specifica la porta di rete a cui connettersi sul server
  Restituisce: un oggetto di tipo "connessione SMTP"

- ***`c`*****`.starttls()`**
  Questo metodo crea la connessione con il server di posta elettronica rappresentato da *c* e attiva la comunicazione sicura.

- ***`c`*****`.login(`***`nomeUtente, password`***`)`**
  Questo metodo esegue l'accesso sul server usando le credenziali fornite.
  Parametri: *nomeUtente*, una stringa contenente il nome dell'utente
  password, una stringa contenente la password dell'utente

- ***`c`*****`.send_message(`***`messaggio`***`)`**
  Questo metodo trasmette un *messaggio* di posta elettronica dal vostro computer al server rappresentato da *c*, il quale, a sua volta, inoltra il messaggio al destinatario specificato.
  Parametro: *messaggio*, un oggetto di tipo `Message`, con intestazione e corpo

- ***`c`*****`.quit()`**
  Questo metodo chiude la comunicazione con il server rappresentato da *c*.

Il modulo `sys`

- **`argv`**
  Questa variabile fa riferimento a una lista di stringhe che memorizza gli argomenti usati sulla riga di comando per mettere in esecuzione il programma.

- **`exit()`**
- **`exit(`***`messaggio`***`)`**
  Questa funzione termina l'esecuzione del programma. Se viene fornito un argomento, prima della terminazione viene visualizzato il *messaggio*.
  Parametro: *messaggio*, una stringa da visualizzare sulla finestra di console

Il modulo `time`

- **`sleep(`***`secondi`***`)`**
  Questa funzione sospende l'esecuzione del programma per un dato numero di *secondi*.
  Parametro: *secondi*, il numero di secondi di sospensione del programma

- **`time()`**
  Questa funzione restituisce la differenza di tempo, misurata in secondi, tra l'istante attuale e la mezzanotte del giorno 1 gennaio 1970, secondo il Tempo Universale (*Universal Time*).
  Restituisce: la differenza di tempo, in secondi, tra ora e la mezzanotte del 1 gennaio 1970

Il modulo `urllib.parse`

- **`urlencode(`***`parametri`***`)`**
  Questa funzione crea una stringa che contiene una sequenza di argomenti associati a un URL, nel formato adatto all'utilizzo come argomento della funzione `urllib.request.urlopen`. Gli argomenti da inserire nella stringa vanno specificati come coppie chiave/valore nel dizionario *parametri*.
  Parametro: *parametri*, un dizionario contenente le coppie chiave/valore che descrivono gli argomenti dell'URL
  Restituisce: una stringa contenente gli argomenti associati a un URL, nel formato corretto

Il modulo `urllib.request`

- *`r`* `=` **`urlopen(`***`url`***`)`**
  Questa funzione si connette alla pagina web identificata dall'indirizzo *url* e la apre in modalità di lettura, restituendo un oggetto di tipo `HTTPResponse` che gestirà le operazioni di lettura.
  Parametro: *url*, una stringa contenente l'indirizzo in formato URL della pagina web da leggere
  Restituisce: un oggetto di tipo `HTTPResponse`

Classe `HTTPResponse`

- `r.read()`
  Questo metodo legge il corpo della pagina web rappresentata dall'oggetto *r* di tipo `HTTPResponse` e la restituisce sotto forma di sequenza di byte.
  Restituisce: una sequenza di byte contenente l'intero corpo della pagina web

- `r.close()`
  Questo metodo chiude la connessione rappresentata dall'oggetto *r* di tipo `HTTPResponse`.

Il modulo `ezgraphics`

Il modulo `ezgraphics` utilizzato in questo libro è basato sui componenti grafici di un progetto *open source*. Visitate il sito `http://ezgraphics.org` per avere maggiori informazioni sul progetto e per consultare la documentazione più completa sull'utilizzo di tutte le caratteristiche disponibili.

Classe `GraphicsWindow`

- `w = GraphicsWindow()`
- `w = GraphicsWindow(larghezza, altezza)`
  Questa funzione crea una nuova finestra che contiene un pannello grafico (*canvas*) vuoto. La dimensione del pannello è 400 per 400 pixel, a meno che non siano definiti i parametri *larghezza* e *altezza*.
  Parametri: *larghezza*, *altezza*, le dimensioni del pannello grafico contenuto nella finestra, in pixel
  Restituisce: il nuovo oggetto di tipo "finestra con pannello grafico"

- `w.canvas()`
  Questo metodo restituisce un riferimento al pannello grafico contenuto nella finestra *w*.
  Restituisce: un riferimento al pannello grafico

- `w.close()`
  Questo metodo chiude la finestra *w* e la elimina definitivamente dal *desktop*. Dopo essere stata chiusa, una finestra non può più essere utilizzata.

- `w.getKey()`
  Questo metodo acquisisce e restituisce il successivo tasto premuto dall'utente sulla tastiera. Il programma sospende la propria esecuzione e rimane in attesa della pressione di un tasto.
  Restituisce: una stringa che indica quale tasto è stato premuto

- `w.getMouse()`
  Questo metodo acquisisce e restituisce la posizione del mouse nel momento in cui ne viene premuto il pulsante all'interno del pannello grafico di *w*. Il programma

sospende la propria esecuzione e rimane in attesa della pressione del tasto del mouse all'interno del pannello grafico di *w*.

Restituisce: una tupla di due elementi, (*x*, *y*), contenente le coordinate del punto del pannello grafico in cui è stato premuto il pulsante del mouse

- **`w.hide()`**
  Questo metodo nasconde la finestra *w*. La finestra rimane aperta e utilizzabile, ma non è visibile sul *desktop*. Se la finestra non è valida, viene sollevata un'eccezione.

- **`w.isValid()`**
  Questo metodo determina se la finestra *w* è valida (cioè se è aperta).
  Restituisce: `True` se la finestra è valida (cioè aperta), altrimenti, se è chiusa, `False`

- **`w.setTitle(`*`titolo`*`)`**
  Questo metodo imposta il testo che viene visualizzato nella barra del titolo della finestra *w*.
  Parametro: *titolo*, la stringa che sarà il titolo della finestra

- **`w.show()`**
  Questo metodo rende visibile la finestra *w* sul *desktop*. Se la finestra non è valida, viene sollevata un'eccezione.

- **`w.wait()`**
  Questo metodo tiene aperta la finestra *w* e attende che l'utente selezioni il pulsante di chiusura presente sulla barra del titolo oppure che il programma invochi la funzione `close`.

Classe `GraphicsCanvas`

- **`c.clear()`**
  Questo metodo ripulisce il pannello grafico *c* rimuovendo tutte le forme geometriche e il testo che vi sono stati precedentemente disegnati.

- **`c.drawArc(`*`x`*`, `*`y`*`, `*`diametro`*`, `*`angoloIniziale`*`, `*`estensione`*`)`**
  Questo metodo disegna un arco circolare nel pannello *c*. Lo stile usato per disegnare l'arco è stato specificato in precedenza usando il metodo `setArcStyle` e lo stile standard (`"slice"`) disegna un settore circolare (*pie slice*, letteralmente "fetta di torta").

  Parametri: *x*, *y*, le coordinate del vertice superiore sinistro del rettangolo di delimitazione dell'arco
  diametro, il diametro (espresso in pixel da un numero intero) del cerchio del quale l'arco fa parte
  angoloIniziale, l'angolo (espresso in gradi) al centro del cerchio che indica il punto iniziale dell'arco (un angolo uguale a zero indica il punto di intersezione tra la circonferenza e la parte destra dell'asse *x* del sistema di coordinate cartesiane avente origine nel centro del cerchio)
  estensione, la dimensione dell'arco fornita come angolo, espresso in gradi

Restituisce: un numero intero che identifica in modo univoco l'oggetto (arco, corda o settore circolare) disegnato nel pannello

- `c.drawImage(`*`immagine`*`)`
- `c.drawImage(`*`x, y, immagine`*`)`

  Questo metodo disegna un'*immagine* nel pannello *c*. L'immagine viene disegnata in modo che il suo angolo superiore sinistro coincida con il punto di coordinate (0, 0) o (*x*, *y*), se questi parametri sono specificati. Nel primo caso, il pannello viene ridimensionato in modo da coincidere con il rettangolo di delimitazione dell'immagine.

  Parametri: *x*, *y*, le coordinate del vertice superiore sinistro dell'immagine
  immagine, un oggetto di tipo `GraphicsImage` contenente l'immagine da visualizzare

  Restituisce: un numero intero che identifica in modo univoco l'immagine disegnata nel pannello

- `c.drawLine(`x_1`, `y_1`, `x_2`, `y_2`)`

  Questo metodo disegna nel pannello *c* un segmento tra i due punti specificati.

  Parametri: x_1, y_1, le coordinate del punto iniziale (due numeri interi)
  x_2, y_2, le coordinate del punto finale (due numeri interi)

  Restituisce: un numero intero che identifica in modo univoco il segmento disegnato nel pannello

- `c.drawOval(`*`x, y, larghezza, altezza`*`)`

  Questo metodo disegna un ovale nel pannello *c*.

  Parametri: *x*, *y*, le coordinate del vertice superiore sinistro del rettangolo di delimitazione
  larghezza, *altezza*, le dimensioni del rettangolo di delimitazione

  Restituisce: un numero intero che identifica in modo univoco l'ovale disegnato nel pannello

- `c.drawPoint(`*`x, y`*`)`

  Questo metodo disegna un punto nel pannello *c*.

  Parametri: *x*, *y*, le coordinate del punto (due numeri interi)

  Restituisce: un numero intero che identifica in modo univoco il punto disegnato nel pannello

- `c.drawPoly(`*`sequenza`*`)`
- `c.drawPoly(`x_1`, `y_1`, `x_2`, `y_2`, `x_3`, `y_3`, . . .)`

  Questo metodo disegna nel pannello *c* un poligono, usando i vertici specificati come una *sequenza* di coordinate *x* e *y*. In alternativa, le coordinate dei vertici possono essere specificate come un elenco di più argomenti.

  Parametri: *sequenza*, una lista o tupla di numeri interi che specificano le coordinate dei vertici del poligono (deve contenere almeno sei valori)
  x_1, y_1, x_2, y_2, x_3, y_3, …, le coordinate dei vertici del poligono (numeri interi)

  Restituisce: un numero intero che identifica in modo univoco il poligono disegnato nel pannello

- **`c.drawRect(x, y, larghezza, altezza)`**
  Questo metodo disegna un rettangolo nel pannello *c*.
  Parametri: *x*, *y*, le coordinate del vertice superiore sinistro del rettangolo
  larghezza, *altezza*, larghezza e altezza del rettangolo
  Restituisce: un numero intero che identifica in modo univoco il rettangolo disegnato nel pannello

- **`c.drawText(x, y, testo)`**
  Questo metodo disegna una stringa di testo nel pannello *c*. Il *testo* viene disegnato in posizione relativa al punto di ancoraggio, di coordinate (*x*, *y*), posizionato nel vertice superiore sinistro del rettangolo che delimita il testo (la posizione del punto di ancoraggio rispetto al testo può essere modificata con il metodo `setAnchor`). Se la stringa contiene dei caratteri *newline* (cioè `\n`), vengono scritte più righe di testo.
  Parametri: *x*, *y*, le coordinate del punto di ancoraggio
  testo, la stringa contenente il testo da disegnare
  Restituisce: un numero intero che identifica in modo univoco il testo disegnato nel pannello

- **`c.drawVector(`x_1`, `y_1`, `x_2`, `y_2`)`**
  Questo metodo disegna nel pannello *c*, tra i due punti specificati, un segmento che termina con una freccia (cioè un "vettore").
  Parametri: x_1, y_1, le coordinate del punto iniziale (due numeri interi)
  x_2, y_2, le coordinate del punto finale (due numeri interi)
  Restituisce: un numero intero che identifica in modo univoco il vettore disegnato nel pannello

- **`c.height()`**
  Questo metodo restituisce l'altezza (cioè la dimensione verticale) del pannello *c*.
  Restituisce: l'altezza del pannello

- **`c.width()`**
  Questo metodo restituisce la larghezza (cioè la dimensione orizzontale) del pannello *c*.
  Restituisce: la larghezza del pannello

- **`c.setAnchor(posizione)`**
  Questo metodo imposta la posizione di ancoraggio che verrà usata per disegnare testo nel pannello *c*. La posizione è un punto del rettangolo che delimita il testo e viene specificata con una direzione "geografica". Infatti, la *posizione* può essere una delle stringhe seguenti: `"n"` (nord), `"s"` (sud), `"e"` (est), `"w"` (ovest), `"nw"` (nord-ovest), `"ne"` (nord-est), `"sw"` (sud-ovest), `"se"` (sud-est), `"center"` (centro).
  Parametro: *posizione*, la posizione di ancoraggio (una delle stringhe elencate)

- **`c.setArcStyle(stile)`**
  Questo metodo imposta lo stile che verrà usato per disegnare archi di cerchio nel pannello *c*. Un arco può essere disegnato in tre modi: come settore circolare (`"slice"`), come corda (`"chord"`) o come arco vero e proprio (`"arc"`).
  Parametro: *stile*, lo stile per disegnare archi (una delle stringhe elencate)

- **`c.setBackground(`*`nome`*`)`**
- **`c.setBackground(`*`rosso, verde, blu`*`)`**

  Questo metodo imposta il colore dello sfondo del pannello *c*. Il colore può essere specificato per *nome* o usando i valori delle sue componenti: *rosso*, *verde* e *blu*.

  Parametri: *nome*, la stringa che indica il nome di un colore
  rosso, *verde*, *blu*, numeri interi compresi tra 0 e 255 (estremi inclusi)

- **`c.setColor(`*`nome`*`)`**
- **`c.setColor(`*`rosso, verde, blu`*`)`**

  Questo metodo imposta allo stesso valore tanto il colore di riempimento ("fill") quanto il colore del profilo ("outline") che verranno usati per disegnare forme nel pannello *c*. Il colore può essere specificato per *nome* o usando i valori delle sue componenti: *rosso*, *verde* e *blu*.

  Parametri: *nome*, la stringa che indica il nome di un colore
  rosso, *verde*, *blu*, numeri interi compresi tra 0 e 255 (estremi inclusi)

- **`c.setFill()`**
- **`c.setFill(`*`nome`*`)`**
- **`c.setFill(`*`rosso, verde, blu`*`)`**

  Questo metodo imposta il colore di riempimento che verrà usato per disegnare forme nel pannello *c*. Il colore può essere specificato per *nome* o usando i valori delle sue componenti: *rosso*, *verde* e *blu*. Se non viene specificato alcun argomento, non verrà usato alcun colore di riempimento.

  Parametri: *nome*, la stringa che indica il nome di un colore
  rosso, *verde*, *blu*, numeri interi compresi tra 0 e 255 (estremi inclusi)

- **`c.setFont(`*`famiglia, stile, dimensione`*`)`**

  Questo metodo imposta il font di caratteri che verrà usato per disegnare testo nel pannello *c*. Un font viene specificato da tre caratteristiche: *famiglia* (scelta tra `"arial"`, `"courier"`, `"times"` e `"helvetica"`), *stile* (scelto tra `"normal"`, `"bold"`, cioè grassetto, `"italic"`, cioè corsivo, e `"bold italic"`) e *dimensione* (misurata in punti tipografici o, semplicemente, punti).

  Parametri: *famiglia*, la stringa che indica il nome del font
  stile, la stringa che indica lo stile del font
  dimensione, un numero intero positivo che indica la dimensione del font

- **`c.setHeight(`*`dimensione`*`)`**

  Questo metodo modifica l'altezza (cioè la dimensione verticale) del pannello *c*.

  Parametro: *dimensione*, un numero intero positivo che indica la nuova dimensione

- **`c.setJustify(`*`stile`*`)`**

  Questo metodo imposta lo stile di giustificazione dei margini (scelto tra `"left"`, cioè incolonnato a sinistra, `"right"`, cioè incolonnato a destra, e `"center"`, cioè centrato orizzontalmente) che verrà usato per disegnare più righe di testo nel pannello *c*.

  Parametro: *stile*, la stringa che indica lo stile di giustificazione dei margini

- **`c.setLineWidth(`*`dimensione`*`)`**
  Questo metodo imposta lo spessore delle linee che verranno disegnate dal pannello *c*.
  Parametro: *dimensione*, un numero intero non negativo

- **`c.setLineStyle(`*`stile`*`)`**
  Questo metodo imposta lo stile (scelto tra `"solid"`, cioè linea continua, e `"dashed"`, cioè linea tratteggiata) che verrà usato per disegnare linee nel pannello *c*.
  Parametro: *stile*, la stringa che indica lo stile di tratteggio delle linee

- **`c.setOutline()`**
- **`c.setOutline(`*`nome`*`)`**
- **`c.setOutline(`*`rosso, verde, blu`*`)`**
  Questo metodo imposta il colore del profilo che verrà usato per disegnare forme nel pannello *c*. Il colore può essere specificato per *nome* o usando i valori delle sue componenti: *rosso*, *verde* e *blu*. Se non viene specificato alcun argomento, i profili non saranno colorati.
  Parametri: *nome*, la stringa che indica il nome di un colore
  rosso, *verde*, *blu*, numeri interi compresi tra 0 e 255 (estremi inclusi)

- **`c.setWidth(`*`dimensione`*`)`**
  Questo metodo modifica la larghezza (cioè la dimensione orizzontale) del pannello *c*.
  Parametro: *dimensione*, un numero intero positivo che indica la nuova dimensione

Classe `GraphicsImage`

- **`i = GraphicsImage(`*`nomeFile`*`)`**
- **`i = GraphicsImage(`*`larghezza, altezza`*`)`**
  Questa funzione crea un nuovo oggetto che può ospitare un'immagine in formato RGB. L'immagine viene caricata leggendo il file *nomeFile* (deducendone le dimensioni) oppure viene creata vuota, con dimensioni uguali a *larghezza* e *altezza*.
  Parametri: *nomeFile*, una stringa che contiene il nome di un file in formato GIF o PPM
  larghezza, *altezza*, le dimensioni della nuova immagine vuota
  Restituisce: il nuovo oggetto di tipo "immagine"

- **`i.clear()`**
  Questo metodo cancella l'intera immagine *i* e rende trasparenti tutti i suoi pixel, senza modificare le dimensioni dell'immagine stessa.

- **`i.copy()`**
  Questo metodo crea una copia dell'immagine *i*.
  Restituisce: il nuovo oggetto di tipo "immagine", duplicato di quello esistente

- **`i.getPixel(`*`riga, colonna`*`)`**
  Questa funzione restituisce il colore di uno specifico pixel dell'immagine *i*. Il colore viene memorizzato in una tupla contenente i valori delle sue componenti, nell'ordine: rosso, verde e blu.
  Parametri: *riga*, *colonna*, le coordinate, orizzontale e verticale, del pixel

Restituisce: una tupla contenente i valori delle componenti di colore

- `i.getRed(riga, colonna)`
- `i.getBlue(riga, colonna)`
- `i.getGreen(riga, colonna)`
  Questa funzione restituisce la corrispondente componente di colore di uno specifico pixel dell'immagine *i*..
  Parametri: *riga*, *colonna*, le coordinate, orizzontale e verticale, del pixel
  Restituisce: la componente di colore richiesta (un numero intero)

- `i.height()`
  Questo metodo restituisce l'altezza (cioè la dimensione verticale) dell'immagine *i*.
  Restituisce: l'altezza dell'immagine

- `i.save(nomeFile)`
  Questo metodo archivia nel file *nomeFile* una copia dell'immagine *i*, in formato GIF.
  Parametro: *nomeFile*, una stringa che contiene il nome del file da scrivere

- `i.setPixel(riga, colonna, colore)`
- `i.setPixel(riga, colonna, rosso, verde, blu)`
  Questo metodo imposta il colore di uno specifico pixel dell'immagine *i*. Il colore può essere descritto dalle sue componenti di colore (*rosso*, *verde* e *blu*) oppure da una tupla contenente quegli stessi valori.
  Parametri: *riga*, *colonna*, le coordinate, orizzontale e verticale, del pixel
  colore, una tupla contenente tre numeri interi compresi tra 0 e 255 (estremi inclusi)
  rosso, *verde*, *blu*, numeri interi compresi tra 0 e 255 (estremi inclusi)

- `i.width()`
  Questo metodo restituisce la larghezza (cioè la dimensione orizzontale) dell'immagine *i*.
  Restituisce: la larghezza dell'immagine

Il modulo `turtle`

- `backward(distanza)`
  Questa funzione fa percorrere alla tartaruga la *distanza* specificata in linea retta, nella direzione opposta a quella verso cui è rivolta.
  Parametro: *distanza*, la distanza da percorrere, in pixel

- `clear()`
  Questa funzione cancella tutto quanto disegnato dalla tartaruga, senza spostarla.

- `forward(distanza)`
  Questa funzione fa percorrere alla tartaruga la *distanza* specificata in linea retta, nella direzione verso cui è rivolta.

Parametro: *distanza*, la distanza da percorrere, in pixel

- **`goto(`***`x, y`***`)`**
  Questa funzione porta la tartaruga nella posizione di coordinate (*x*, *y*).
  Parametri: *x*, *y*, le coordinate di una posizione nella finestra (due numeri interi)

- **`heading()`**
  Questa funzione restituisce l'angolo che descrive la direzione verso cui è rivolta la tartaruga.
  Restituisce: la direzione verso cui è rivolta la tartaruga (un numero in virgola mobile)

- **`hideturtle()`**
  Questa funzione rende invisibile la tartaruga, i cui successivi spostamenti non disegneranno nella finestra.

- **`home()`**
  Questa funzione porta la tartaruga nell'origine delle coordinate, (0, 0), e la orienta verso est.

- **`left(`***`angolo`***`)`**
  Questa funzione ruota la tartaruga verso "sinistra" (cioè in senso antiorario) dell'*angolo* indicato.
  Parametro: *angolo*, l'angolo di cui ruotare (un numero in virgola mobile)

- **`pencolor(`***`nome`***`)`**
- **`pencolor(`***`rosso, verde, blu`***`)`**
  Questa funzione imposta il colore della penna usata dalla tartaruga per disegnare. Il colore può essere specificato per *nome* oppure descritto dalle sue componenti: *rosso*, *verde* e *blu*.
  Parametri: *nome*, una stringa che indica il nome di un colore
  rosso, *verde*, *blu*, numeri interi compresi tra 0 e 255 (estremi inclusi)

- **`pendown()`**
  Questa funzione abbassa la penna, in modo che la tartaruga disegni quando si sposta.

- **`pensize(`***`spessore`***`)`**
  Questa funzione imposta lo *spessore* della linea disegnata dalla tartaruga, in pixel.
  Parametro: *spessore*, lo spessore della penna, in pixel (un numero intero)

- **`penup()`**
  Questa funzione alza la penna, in modo che la tartaruga non disegni quando si sposta.

- **`reset()`**
  Questa funzione cancella tutto quanto disegnato dalla tartaruga, porta la tartaruga nell'origine delle coordinate, (0, 0), e la orienta verso est.

- **`right(`*`angolo`*`)`**
  Questa funzione ruota la tartaruga verso "destra" (cioè in senso orario) dell'*angolo* indicato.
  Parametro: *angolo*, l'angolo di cui ruotare (un numero in virgola mobile)

- **`showturtle()`**
  Questa funzione rende visibile la tartaruga, i cui successivi spostamenti disegneranno nella finestra.

D

I sottoinsiemi Basic Latin e Latin-1 di Unicode

Questa appendice elenca i caratteri Unicode usati più spesso nell'elaborazione di lingue dell'Europa occidentale. All'indirizzo `http://unicode.org` è reperibile l'elenco completo dei caratteri Unicode.

Tabella1 Alcuni caratteri di controllo

Carattere	Sequenza di escape	Decimale	Codice
Tab	`"\t"`	9	`"\u0009"`
Nuova riga	`"\n"`	10	`"\u000A"`
Invio	`"\r"`	13	`"\u000D"`
Spazio		32	`"\u0020"`

Tabella 2
Il sottoinsieme Basic Latin (ASCII) di Unicode

Carattere	Codice	Decimale	Carattere	Codice	Decimale
!	"\u0021"	33	P	"\u0050"	80
"	"\u0022"	34	Q	"\u0051"	81
#	"\u0023"	35	R	"\u0052"	82
$	"\u0024"	36	S	"\u0053"	83
%	"\u0025"	37	T	"\u0054"	84
&	"\u0026"	38	U	"\u0055"	85
'	"\u0027"	39	V	"\u0056"	86
(	"\u0028"	40	W	"\u0057"	87
)	"\u0029"	41	X	"\u0058"	88
*	"\u002A"	42	Y	"\u0059"	89
+	"\u002B"	43	Z	"\u005A"	90
,	"\u002C"	44	[	"\u005B"	91
-	"\u002D"	45	\	"\u005C"	92
.	"\u002E"	46	]	"\u005D"	93
/	"\u002F"	47	^	"\u005E"	94
0	"\u0030"	48	_	"\u005F"	95
1	"\u0031"	49	`	"\u0060"	96
2	"\u0032"	50	a	"\u0061"	97
3	"\u0033"	51	b	"\u0062"	98
4	"\u0034"	52	c	"\u0063"	99
5	"\u0035"	53	d	"\u0064"	100
6	"\u0036"	54	e	"\u0065"	101
7	"\u0037"	55	f	"\u0066"	102
8	"\u0038"	56	g	"\u0067"	103
9	"\u0039"	57	h	"\u0068"	104
:	"\u003A"	58	i	"\u0069"	105
;	"\u003B"	59	j	"\u006A"	106
<	"\u003C"	60	k	"\u006B"	107
=	"\u003D"	61	l	"\u006C"	108
>	"\u003E"	62	m	"\u006D"	109
?	"\u003F"	63	n	"\u006E"	110
@	"\u0040"	64	o	"\u006F"	111
A	"\u0041"	65	p	"\u0070"	112
B	"\u0042"	66	q	"\u0071"	113
C	"\u0043"	67	r	"\u0072"	114
D	"\u0044"	68	s	"\u0073"	115
E	"\u0045"	69	t	"\u0074"	116
F	"\u0046"	70	u	"\u0075"	117
G	"\u0047"	71	v	"\u0076"	118
H	"\u0048"	72	w	"\u0077"	119
I	"\u0049"	73	x	"\u0078"	120
J	"\u004A"	74	y	"\u0079"	121
K	"\u004B"	75	z	"\u007A"	122
L	"\u004C"	76	{	"\u007B"	123
M	"\u004D"	77	|	"\u007C"	124
N	"\u004E"	78	}	"\u007D"	125
O	"\u004F"	79	~	"\u007E"	126

Tabella 3
Il sottoinsieme Latin-1 di Unicode

Carattere	Codice	Decimale	Carattere	Codice	Decimale
¡	"\u00A1"	161	Ñ	"\u00D1"	209
¢	"\u00A2"	162	Ò	"\u00D2"	210
£	"\u00A3"	163	Ó	"\u00D3"	211
¤	"\u00A4"	164	Ô	"\u00D4"	212
¥	"\u00A5"	165	Õ	"\u00D5"	213
¦	"\u00A6"	166	Ö	"\u00D6"	214
§	"\u00A7"	167	×	"\u00D7"	215
¨	"\u00A8"	168	Ø	"\u00D8"	216
©	"\u00A9"	169	Ù	"\u00D9"	217
ª	"\u00AA"	170	Ú	"\u00DA"	218
«	"\u00AB"	171	Û	"\u00DB"	219
¬	"\u00AC"	172	Ü	"\u00DC"	220
-	"\u00AD"	173	Ý	"\u00DD"	221
®	"\u00AE"	174	Þ	"\u00DE"	222
¯	"\u00AF"	175	ß	"\u00DF"	223
°	"\u00B0"	176	à	"\u00E0"	224
±	"\u00B1"	177	á	"\u00E1"	225
²	"\u00B2"	178	â	"\u00E2"	226
³	"\u00B3"	179	ã	"\u00E3"	227
´	"\u00B4"	180	ä	"\u00E4"	228
µ	"\u00B5"	181	å	"\u00E5"	229
¶	"\u00B6"	182	æ	"\u00E6"	230
·	"\u00B7"	183	ç	"\u00E7"	231
¸	"\u00B8"	184	è	"\u00E8"	232
¹	"\u00B9"	185	é	"\u00E9"	233
º	"\u00BA"	186	ê	"\u00EA"	234
»	"\u00BB"	187	ë	"\u00EB"	235
¼	"\u00BC"	188	ì	"\u00EC"	236
½	"\u00BD"	189	í	"\u00ED"	237
¾	"\u00BE"	190	î	"\u00EE"	238
¿	"\u00BF"	191	ï	"\u00EF"	239
À	"\u00C0"	192	ð	"\u00F0"	240
Á	"\u00C1"	193	ñ	"\u00F1"	241
Â	"\u00C2"	194	ò	"\u00F2"	242
Ã	"\u00C3"	195	ó	"\u00F3"	243
Ä	"\u00C4"	196	ô	"\u00F4"	244
Å	"\u00C5"	197	õ	"\u00F5"	245
Æ	"\u00C6"	198	ö	"\u00F6"	246
Ç	"\u00C7"	199	÷	"\u00F7"	247
È	"\u00C8"	200	ø	"\u00F8"	248
É	"\u00C9"	201	ù	"\u00F9"	249
Ê	"\u00CA"	202	ú	"\u00FA"	250
Ë	"\u00CB"	203	û	"\u00FB"	251
Ì	"\u00CC"	204	ü	"\u00FC"	252
Í	"\u00CD"	205	ý	"\u00FD"	253
Î	"\u00CE"	206	þ	"\u00FE"	254
Ï	"\u00CF"	207	ÿ	"\u00FF"	255
Ð	"\u00D0"	208			

E

Numeri binari e operazioni tra bit

Numeri binari

La notazione decimale rappresenta i numeri come potenze di 10, ad esempio:

$$1729_{\text{decimale}} = 1 \times 10^3 + 7 \times 10^2 + 2 \times 10^1 + 9 \times 10^0$$

Non c'è nessun motivo particolare per la scelta del numero 10, a parte il fatto che, storicamente, molti sistemi di numerazione sono stati messi a punto da persone che contavano con le dita delle mani, che sono dieci. Altri sistemi numerici, con base 12, 20 o 60, sono stati usati da varie culture nel corso della storia dell'umanità, tuttavia i computer usano un sistema numerico con base 2, perché è molto più facile costruire componenti elettronici che funzionino con due valori, che possono essere rappresentati da una corrente che scorre oppure no, piuttosto che rappresentare 10 valori diversi di un segnale elettrico. Un numero scritto in base 2 viene anche detto numero *binario*. Ad esempio:

$$1101_{\text{binario}} = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 1 = 13$$

Per le cifre che seguono il punto (o la virgola) "decimale", si usano le potenze negative di 2:

$$1.101_{\text{binario}} = 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 1 + 1/2 + 1/8 = 1 + 0.5 + 0.125 = 1.625$$

In generale, per convertire un numero binario nel suo equivalente decimale, si valutano e si sommano, semplicemente, le potenze di 2 che corrispondono alle cifre di valore 1. La Tabella 1 mostra le prime potenze di 2.

Tabella 1
Potenze di due

Potenza	Valore decimale
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024
2^{11}	2048
2^{12}	4096
2^{13}	8192
2^{14}	16384
2^{15}	32768
2^{16}	65536

Per convertire in binario un numero intero decimale, lo si divide ripetutamente per 2, tenendo traccia dei resti e fermandosi quando il dividendo diventa 0. Quindi, iniziando dall'*ultimo*, si scrivono i resti sotto forma di numero binario. Ad esempio:

$$\begin{aligned} 100 \div 2 &= 50 \text{ resto } \mathbf{0} \\ 50 \div 2 &= 25 \text{ resto } \mathbf{0} \\ 25 \div 2 &= 12 \text{ resto } \mathbf{1} \\ 12 \div 2 &= 6 \text{ resto } \mathbf{0} \\ 6 \div 2 &= 3 \text{ resto } \mathbf{0} \\ 3 \div 2 &= 1 \text{ resto } \mathbf{1} \\ 1 \div 2 &= 0 \text{ resto } \mathbf{1} \end{aligned}$$

Quindi, $100_{\text{decimale}} = 1100100_{\text{binario}}$.

Per convertire, invece, in binario un numero frazionario minore di 1, lo si moltiplica ripetutamente per 2: se il risultato è maggiore di 1, si sottrae 1 e ci si ferma quando il numero da moltiplicare diventa 0. Quindi, iniziando dalla *prima*, si scrivono le cifre che precedono il punto decimale come cifre binarie della parte frazionaria. Ad esempio:

$$\begin{aligned} 0.35 \cdot 2 &= \mathbf{0}.7 \\ 0.7 \cdot 2 &= \mathbf{1}.4 \\ 0.4 \cdot 2 &= \mathbf{0}.8 \\ 0.8 \cdot 2 &= \mathbf{1}.6 \\ 0.6 \cdot 2 &= \mathbf{1}.2 \\ 0.2 \cdot 2 &= \mathbf{0}.4 \end{aligned}$$

A questo punto lo schema si ripete, quindi la rappresentazione binaria di 0.35 è 0.01 0110 0110 0110 ...

Per convertire in binario un numero in virgola mobile, si convertono, separatamente, la sua parte intera e la sua parte frazionaria.

Numeri interi in complemento a due

Python rappresenta i numeri interi negativi con la notazione "in complemento a due" (*two's complement*). Per costruire la rappresentazione di un numero intero negativo a partire dalla rappresentazione del suo valore assoluto si procede così:

- Si cambia valore a ciascun bit (operazione detta *flip*).
- Si aggiunge 1 al risultato ottenuto.

Ad esempio, per rappresentare il valore –13 con 8 bit, per prima cosa si cambia il valore di tutti i bit della rappresentazione binaria di 13, che è 00001101, ottenendo 11110010, poi si aggiunge 1, per cui –13 = 11110011.

Per sommare numeri interi, alcuni dei quali possono essere negativi, non serve alcuna regola speciale: semplicemente, si seguono le normali regole dell'addizione in colonna, con il riporto verso la colonna successiva nel caso in cui la somma delle cifre e del riporto precedente sia 2 o 3. Ad esempio:

```
      1 1111 111
+13     0000 1101
-13     1111 0011
      1 0000 0000
```

Sono importanti, però, soltanto gli ultimi 8 bit del risultato, per cui +13 e –13 hanno somma 0, come dovrebbero.

In particolare, la rappresentazione in complemento a due di –1 è 1111...1111, cioè tutti i suoi bit valgono 1.

Il bit più a sinistra di un numero in complemento a due vale 0 se il numero è positivo oppure zero e 1 se è negativo.

Operazioni tra bit e scorrimenti

In Python esistono quattro operazioni che agiscono su bit: la *negazione unaria* (`~`) e le operazioni binarie *and* (`&`), *or* (`|`) e *or esclusivo* (`^`), detto anche *xor* (che sta per "exclusive *or*").

Le Tabelle 2 e 3 sono le cosiddette "tabelle di verità" per le operazioni sui bit in Python. Quando un'operazione sui bit viene applicata a numeri interi, l'operazione viene svolta sui bit corrispondenti.

Tabella 2
L'operazione di negazione unaria

a	~a
0	1
1	0

Tabella 3
Le operazioni binarie and, or e xor

a	b	a&b	a|b	a^b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Ad esempio, supponete di voler calcolare `46 & 13`. Per prima cosa convertite in binario entrambi i valori: $46_{\text{decimale}} = 101110_{\text{binario}}$ (in realtà, essendo un intero a 32 bit, 00000000000000000000000101110) e $13_{\text{decimale}} = 1101_{\text{binario}}$. Poi, effettuate l'operazione sui bit corrispondenti:

```
  0.....0101110
& 0.....0001101

  0.....0001100
```

La risposta è $1100_{\text{binario}} = 12_{\text{decimale}}$.

A volte si vede l'operatore ¦ usato per combinare due schemi di bit. Se, ad esempio, la costante `BOLD` (*grassetto*) ha il valore 1, e la costante `ITALIC` (*corsivo*) ha il valore 2, la loro combinazione binaria *or*, cioè `BOLD ¦ ITALIC`, ha impostati a 1 tanto il bit per il grassetto quanto quello per il corsivo:

```
  0.....0000001
¦ 0.....0000010

  0.....0000011
```

Oltre alle operazioni che agiscono su singoli bit, esistono anche due operazioni di *scorrimento* ("shift") che prendono lo schema di bit di un numero e lo spostano a sinistra o a destra di un dato numero di posizioni.

Lo scorrimento a sinistra (*left shift*, che usa come simbolo `<<`) sposta tutti i bit verso sinistra, inserendo zeri nei bit meno significativi. Lo spostamento a sinistra di n bit fornisce lo stesso risultato di una moltiplicazione per 2^n. Lo scorrimento verso destra (*right shift*, che usa come simbolo `>>`) sposta tutti i bit a destra, *propagando* il bit di segno: quindi, il risultato è uguale a quello della divisione intera per 2^n, sia per valori positivi che per valori negativi.

L'espressione:

```
1 << n
```

genera uno schema di bit in cui il solo bit `n`-esimo vale 1 (contando le posizioni a partire dalla posizione 0 del bit meno significativo).

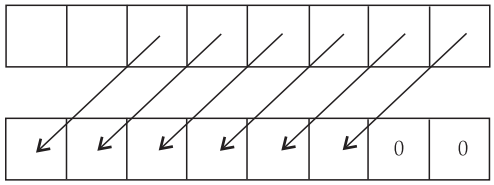
Per impostare a 1 il bit `n`-esimo di un numero, quindi, si può eseguire l'operazione:

```
x = x ¦ 1 << n
```

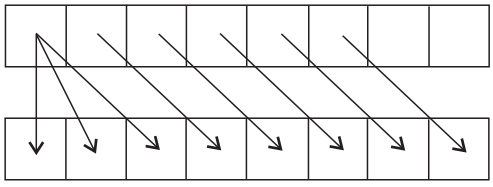
Per controllare se il bit `n`-esimo di un numero vale 1, si può eseguire la verifica:

```
if (x & 1 << n) != 0 :
```

Figura 1
Le operazioni di scorrimento



Scorrimento a sinistra (<<)



Scorrimento a destra con estensione del segno (>>)

F

Compendio di HTML

Introduzione al linguaggio HTML

Le pagine web sono scritte in linguaggio HTML (HyperText Markup Language) e, come il codice Python, il codice HTML è costituito da testo che segue alcune rigide regole grammaticali. Quando un browser web legge una pagina, ne *interpreta* il codice e la *visualizza*, con caratteri, paragrafi, tabelle e immagini.

I file HTML contengono testo e *marcatori* (*tag*) che danno istruzioni al browser sulle modalità di visualizzazione del testo. Nella versione attuale, i marcatori sono dozzine e quelli più importanti sono riportati nella Tabella 1: per vostra fortuna, per iniziare vi basta conoscerne soltanto alcuni. Quasi tutti i marcatori HTML vengono usati a coppie, con un marcatore di apertura e uno di chiusura: ciascuna coppia si applica al testo che si trova tra i due marcatori corrispondenti. Ecco un esempio tipico:

```
Python is a <i>high-level</i> programming language.
```

La coppia di marcatori `<i>` e `</i>` indica al browser di visualizzare *in corsivo* ("italics", da cui la *i* usata nel marcatore) il testo presente all'interno:

Python is a *high-level* programming language.

Il marcatore di chiusura è identico a quello di apertura corrispondente, tranne per il fatto che al nome del marcatore viene prefisso un carattere `/`. Ad esempio, il testo in grassetto (*bold-face*) è delimitato dai marcatori `<b>` e `</b>`, mentre un paragrafo viene inserito all'interno della coppia `<p>` e `</p>`.

```
<p><b>Python</b> is a <i>high-level</i> programming language.</p>
```

Tabella 1: I marcatori HTML più comuni

Marcatore	Significato	Figli	Attributi più comuni
`html`	Documento HTML	`head, body`	
`head`	Intestazione del documento	`title`	
`title`	Titolo del documento		
`body`	Corpo del documento		
`h1 . . . h6`	Intestazioni di livello 1 … 6		
`p`	Paragrafo		
`ul`	Elenco puntato	`li`	
`ol`	Elenco numerato	`li`	
`dl`	Elenco di definizioni	`dt, dd`	
`li`	Elemento di elenco		
`dt`	Termine da definire		
`dd`	Definizione di termine		
`table`	Tabella	`tr`	
`tr`	Riga di tabella	`th, td`	
`th`	Elemento di intestazione in tabella		
`td`	Elemento di dato in tabella		
`a`	Punto di riferimento		`href, name`
`img`	Immagine		`src, width, height`
`pre`	Testo pre-impaginato		
`hr`	Riga orizzontale		
`br`	A capo		
`i` oppure `em`	Font corsivo		
`b` oppure `strong`	Font grassetto		
`tt` oppure `code`	Font monospaziato		
`s` oppure `strike`	Font barrato		
`u`	Font sottolineato		
`super`	Font in stile "apice"		
`sub`	Font in stile "pedice"		
`form`	Modulo da riempire		`action, method`
`input`	Campo di inserimento dati		`type, name, value, size, checked`
`select`	Campo di selezione	`Option`	`name`
`option`	Opzione per selezione		
`textarea`	Zona di testo con più righe		`name, rows, cols`

Il risultato è il paragrafo seguente:

Python is a *high-level* programming language.

Un'altra struttura sintattica molto utilizzata è l'elenco puntato (*bulleted list*). Ad esempio:

Python is

- easy to learn
- portable
- free and open source

Il codice HTML seguente descrive la struttura precedente:

```
<p>Python is</p>
<ul><li>easy to learn</li>
<li>portable</li>
<li>free and open source</li></ul>
```

Ogni elemento dell'elenco è delimitato da una coppia `<li> </li>` (che sta per "list item", cioè, appunto, elemento di un elenco), mentre l'intero elenco, costituito da tutti i suoi elementi, è delimitato dalla coppia `<ul> </ul>` ("unnumbered list", ossia elenco non numerato, nel senso che gli elementi sono introdotti da punti o altri simboli piuttosto che da numeri).

Nel codice HTML si possono inserire liberamente spazi e caratteri per andare a capo (*newline*), per migliorarne la leggibilità. Ad esempio, il codice che abbiamo visto prima, che definisce un elenco puntato, potrebbe essere scritto così:

```
<p>Python is</p>
<ul>
<li>easy to learn</li>
<li>portable</li>
<li>free and open source</li>
</ul>
```

Il browser ignora gli spazi che non si trovano all'interno di marcatori.

Spesso, se un marcatore (come `</li>`) viene omesso, il browser cerca di indovinare il marcatore mancante, anche se a volte fa errori: è sempre meglio scrivere tutti i marcatori previsti.

Nelle pagine web si possono inserire immagini usando il marcatore `img`, che, nella sua forma più semplice, si scrive in questo modo:

```
<img src="hamster.jpeg"/>
```

Questo codice dice al browser di caricare e visualizzare l'immagine memorizzata nel file `hamster.jpeg`. Questo marcatore è un po' diverso dai precedenti: invece di scrivere testo tra la coppia di marcatori `<img>` e `</img>`, si usa un attributo del marcatore `img` per specificare il nome di un file. Gli attributi sono coppie di tipo nome / valore: ad esempio, l'attributo `src` (contrazione di *source*, cioè sorgente) ha `"hamster.jpeg"` come valore. La Tabella 2 presenta gli attributi più utilizzati.

Tabella 2
Gli attributi HTML più comuni

Attributo	Descrizione	Solitamente nel marcatore
`name`	Nome di un punto di riferimento o di un elemento in un modulo	`input, select, textarea, a`
`href`	Riferimento (o collegamento) ipertestuale	`a`
`src`	Sorgente (ad esempio di un'immagine)	`img`
`code`	Codice di un *applet*	`applet`
`width, height`	Larghezza e altezza di un'immagine o di un *applet*	`img,applet`
`rows, cols`	Righe e colonne di una zona di testo	`textarea`
`type`	Tipo di un campo di input, ad esempio `text`, `password`, `checkbox`, `radio`, `submit` o `hidden`	`input`
`value`	Valore di un campo di input o etichetta di un pulsante	`input`
`size`	Dimensione di un campo di testo	`input`
`checked`	Stato di un pulsante di selezione	`input`
`action`	Indirizzo URL dell'azione associata a un modulo	`form`
`method`	`GET` oppure `POST`	`form`

Si ritiene preferibile aggiungere alcuni attributi al marcatore `img`, nello specifico la *dimensione dell'immagine* e una sua *descrizione alternativa*:

```
<img src="hamster.jpeg" width="640" height="480"
alt="A photo of Harry, the Horrible Hamster"/>
```

Questi ulteriori attributi aiutano il browser a disporre geometricamente gli elementi all'interno della pagina e a visualizzare una descrizione temporanea delle immagini fintantoché non abbia terminato di acquisirle (oppure se il browser non è proprio in grado di visualizzare immagini, come accade con un browser vocale destinato a utilizzatori non vedenti): sarà molto apprezzato dagli utenti dotati di una connessione di rete molto lenta.

Dato che non usiamo il marcatore di chiusura `</img>`, inseriamo un carattere `/` prima di chiudere il marcatore di apertura: non è richiesto dal linguaggio HTML, ma è un requisito dello standard XHTML, il linguaggio, basato su XML, che ha preso il posto di HTML (all'indirizzo `www.w3c.org/TR/xhtml1` troverete ulteriori informazioni su XHTML).

Il marcatore più importante nelle pagine web è la coppia `<a> </a>`: il testo presente all'interno diventa un *collegamento* (*link*) a un'altra risorsa, tipicamente un'altra pagina. I collegamenti tra le pagine sono proprio ciò che trasforma il Web in una… ragnatela (in inglese *web*, appunto). Solitamente il browser visualizza i collegamenti in un modo speciale (ad esempio, sottolineando il testo in blu). Ecco il codice di un tipico collegamento:

```
<a href="http://horstmann.com">Cay Horstmann</a> is the author of this book.
```

Quando l'utente che sta guardando la pagina web seleziona con il mouse le parole **Cay Horstmann**, il browser carica la pagina web che si trova all'indirizzo `http://horstmann.com`. Il valore dell'attributo `href` è un URL (*Uniform Resource Locator*) e indica al browser la destinazione corretta: il prefisso `http:` sta per *HyperText Transfer Protocol* e segnala al browser che il file indicato deve essere trattato come una pagina web. Altri protocolli di rete portano ad azioni diverse, come `ftp:` che provoca lo scaricamento del file, `mailto:` che invia un messaggio di posta elettronica e `file:` che visualizza un file presente nel *file system* locale dell'utente.

Come avrete notato, i marcatori sono racchiusi tra "parentesi angolari", che sono anche i simboli usati in matematica per indicare le relazioni "minore di" e "maggiore di". Cosa succede se si vogliono scrivere tali simboli in una pagina web? Il linguaggio HTML mette a disposizione le notazioni `<` (*less than*, cioè "minore di") e `>` (*greater than*, cioè "maggiore di"), che provocano la visualizzazione, rispettivamente, dei simboli < e >. Con una sintassi molto simile, esistono sequenze di codice per rappresentare altri simboli, come le lettere accentate. Il carattere `&` (chiamato *ampersand*, "e commerciale" in italiano) inizia uno di tali codici e `&` rappresenta proprio il carattere &. La Tabella 3 ne riporta alcuni.

Tabella 3
Alcuni simboli speciali in HTML

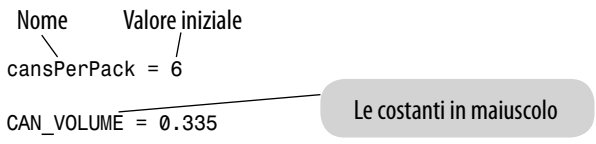
Codice	Descrizione	Simbolo
`<`	Minore di	<
`>`	Maggiore di	>
`&`	E commerciale	&
`&quot;`	Virgolette	"
` `	Spazio ineliminabile	
`&copy;`	Copyright	©

Probabilmente vi sarà già successo di realizzare pagine web usando un *editor web*, che funziona in modo simile a un *word processor* e vi consente di scrivere una pagina web usando la modalità WYSIWYG (*what you see is what you get*, cioè "quello che vedi è quello che otterrai"): anche in quel caso sono comunque presenti i marcatori HTML, che potete vedere caricando il file HTML in un editor di testo. Se vi trovate bene con un editor web di tipo WYSIWYG non avete nessuna necessità di memorizzare i marcatori HTML, ma molti programmatori e professionisti dello sviluppo web preferiscono lavorare direttamente con i marcatori, almeno per rifinire i dettagli delle pagine, perché si ottiene un controllo più preciso e puntuale.

G

Compendio di Python

Definizioni di variabili e costanti



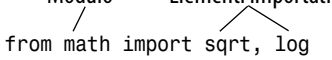
Funzioni matematiche

`abs(x)`	Valore assoluto, $\|x\|$
`round(x)`	Arrotondamento all'intero più vicino
`max(x1, x2, ...)`	Il maggiore tra gli argomenti
`min(x1, x2, ...)`	Il minore tra gli argomenti

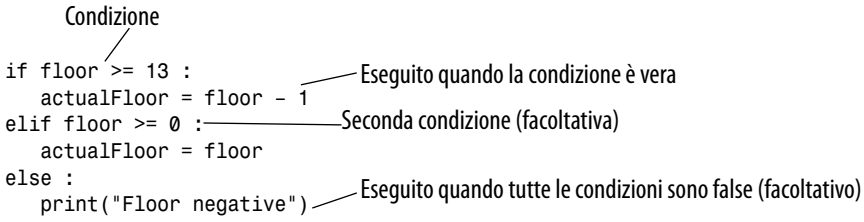
Dal modulo `math`:

`sqrt(x)`	Radice quadrata di x
`trunc(x)`	Troncamento a un intero
`sin(x), cos(x), tan(x)`	Seno, coseno, tangente di x
`degrees(x), radians(x)`	Conversione in gradi e in radianti
`log(x), log(x, base)`	Logaritmo naturale di x e $\log_{base}(x)$

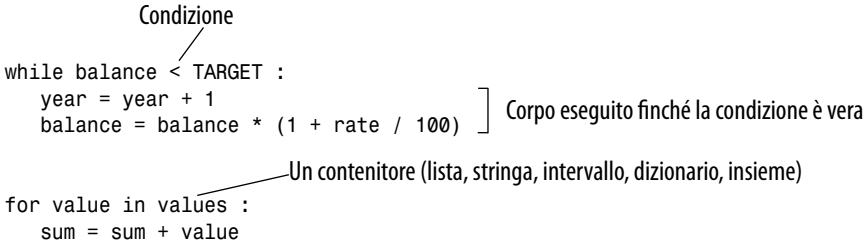
Importazioni



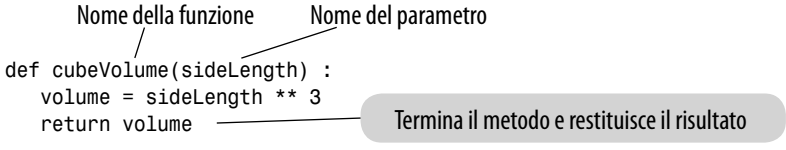
Enunciato condizionale



Enunciati iterativi (cicli)



Definizione di funzione



Operatori di selezione e loro precedenze

(Un elenco completo si trova nell'Appendice A)

`[]`	Accesso a un elemento di una sequenza
`**`	Elevamento a potenza
`* / // %`	Moltiplicazione, divisione, quoziente intero, resto
`+ -`	Addizione, sottrazione
`== < <= > >= != in`	Confronti e appartenenza
`not`	Operatori booleani
`or`	
`and`	

Stringhe

```
s = "Hello"
len(s)                  La lunghezza della stringa: 5
s[1]                    Il carattere che ha indice 1: "e"
s + "!"                 Concatenazione: "Hello!"
s * 2                   Replicazione: "HelloHello"
s.upper()               Restituisce "HELLO"
s.replace("e", "3")     Restituisce "H3llo"
```

Liste

```
friends = []
values = [16, 3, 2, 13]

for i in range(len(values)) :
   values[i] = i * i

friends.append("Bob")
friends.insert(0, "Amy")
if "Amy" in friends :
   n = friends.index("Amy")
   friends.pop(n)
else :
   friends.pop()
friends.remove("Bob")

guests = friends + ["Lee", "Zoe"]
scores = [0] * 12
bestFriends = friends[0 : 3]
total = sum(values)
largest = max(values)
values.sort()
```

Una lista vuota

Elimina l'*n*-esimo elemento

Elimina l'ultimo elemento

Concatenazione

Replicazione

Porzione

Incluso

Escluso

La lista deve contenere numeri

Usando `min` si ottiene il minimo

Tabelle

```
table = [[16, 3, 2, 13],
   [5, 10, 11, 8],
   [9, 6, 7, 12],
   [4, 15, 14, 1]]

for row in range(len(table)) :
   for column in range(len(table[row])) :
      sum = sum + table[row][column]
```

Numero di righe

Numero di colonne

Input e Output

```
name = input("Your name: ")
age = int(input("Your age: "))
weight = float(input("Your weight: "))
print("You are", age, "years old.")

print("Your age: ", end="")
```

Dopo non va a capo

```
print("%-20s Age:%3d %8.2f kg" % (name, age, weight))
```

Ampiezza del campo, Precisione, Stringa allineata a sinistra, Intero, In virgola mobile

Insiemi e dizionari

```
cast = { "Luigi", "Gumbys", "Spiny" }
audience = set()
if "Luigi" in cast :
   cast.remove("Luigi")

contacts = { "Fred": 7235591, "Mary": 3841212 }
oldContacts = {}

contacts["John"] = 4578102
if "Fred" in contacts :
   contacts.pop("Fred")

for key in contacts :
   print(key, contacts[key])
```

Un insieme vuoto

Un dizionario vuoto

Grafica

```
from ezgraphics import GraphicsWindow
win = GraphicsWindow(400, 200)

c = win.canvas()

c.setOutline("red")
c.setFill(255, 0, 100)
c.setColor("gray")

c.drawLine(x1, y1, x2, y2)
```

Larghezza, Altezza

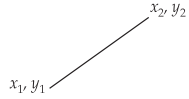
Si disegna nel pannello

Imposta il colore per linee e testo

Imposta il colore per l'interno delle forme

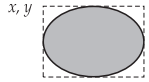
Rosso, Verde, Blu

Imposta il colore per profili e per l'interno



```
c.drawRect(x, y, width, height)
```


```
c.drawOval(x, y, width, height)
```




```
c.drawText(x, y, testo)
```


```
win.wait()
```
Attende che l'utente chiuda la finestra

Elaborazione di file

```
infile = open("input.txt", "r")

for line in infile :
   line = line.rstrip()
   words = line.split()
   fields = line.split(":")

infile.close()
```
Ricordare la chiusura dei file

```
outfile = open("output.txt", "w")
outfile.write("Hello\n")
outfile.write("%10.2f\n" % value)
outfile.close()
```

Definizione di classe

```
class BankAccount :
   def __init__(self, initialBalance) :
      self._balance = initialBalance

   def withdraw(self, amount) :
      self._balance = self._balance - amount

   def getBalance(self) :
      return self._balance

checking = BankAccont(1000)
checking.withdraw(400)
print(checking.getBalance())
```
Costruttore

Variabile di esemplare

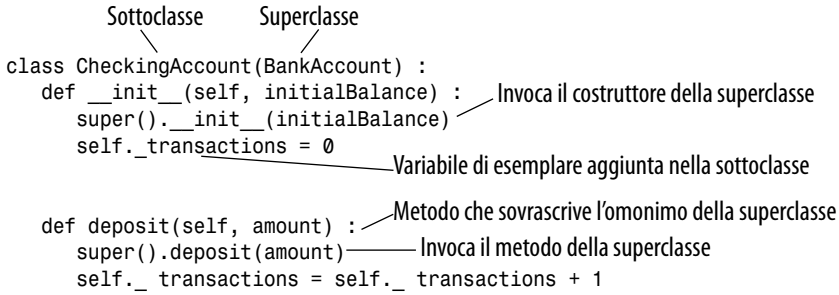
Metodi

Invoca il costruttore

Invoca il metodo

`checking._balance` è privata

Ereditarietà



Eccezioni