Potete inoltre estrarre la cifra più a destra o più cifre di un numero. Ad esempio `x % 10` restituisce la cifra più a destra di `x` (in base 10). E analogamente, `x % 100` restituisce le ultime due cifre.

Per chi usa Python 2, la divisione funziona in modo diverso. L'operatore di divisione intera non esiste, e quello di divisione, `/`, esegue una divisione intera se entrambi gli operandi sono interi, mentre il risultato è un decimale a virgola mobile se almeno uno degli operandi è un decimale.

## 5.2 Espressioni booleane

Un'**espressione booleana** è un'espressione che può essere o vera o falsa. Gli esempi che seguono usano l'operatore `==`, confrontano due valori e restituiscono `True` (vero) se sono uguali, `False` (falso) altrimenti:

```
>>> 5 == 5
True
>>> 5 == 6
False
```

`True` e `False` sono valori speciali che sono di tipo `bool`; non sono delle stringhe:

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

L'operatore `==` è uno degli **operatori di confronto** (chiamati anche operatori relazionali); gli altri sono:

```
x != y               # x è diverso da y
x > y                # x è maggiore di y
x < y                # x è minore di y
x >= y               # x è maggiore o uguale a y
x <= y               # x è minore o uguale a y
```

Queste operazioni vi saranno familiari, tuttavia i simboli usati in Python non sono del tutto uguali a quelli matematici. Un errore frequente è usare il simbolo di uguale(`=`) anziché il doppio uguale (`==`). Ricordatevi la differenza: `=` è un operatore di assegnazione, mentre `==` è un operatore di confronto. Inoltre in Python non esistono simboli come `=<` o `=>`.

## 5.3 Operatori logici

Ci sono tre **operatori logici**: `and`, `or`, e `not`. Il significato di questi operatori è simile a quello comune (e, o, non): per esempio, l'espressione `x > 0 and x < 10` è vera solo se sono vere *entrambe* le condizioni, cioè `x` è più grande di 0 *e* più piccolo di 10.

L'espressione `n % 2 == 0 or n % 3 == 0` invece è vera se è verificata *almeno una* delle due condizioni, cioè se il numero è divisibile per 2 *o* per 3 (o per entrambi).

Infine, l'operatore `not` nega il valore di un'espressione booleana, per cui `not (x > y)` è vera se `x > y` è falsa, cioè se `x` è minore o uguale a `y`.

In senso stretto, gli operandi degli operatori logici dovrebbero essere delle espressioni booleane, ma qui Python non è rigido: ogni numero diverso da zero viene accettato ed interpretato come `True`.

```
>>> 42 and True
True
```

Questa flessibilità può essere utile, ma ci sono alcune sottigliezze che potrebbero confondere. È preferibile evitarla (a meno che non sappiate esattamente quello che state facendo).

## 5.4 Esecuzione condizionale

Se volete scrivere programmi utili, vi capiterà spesso di dover controllare se si verificano determinate condizioni, e di variare di conseguenza il comportamento del programma. Le **istruzioni condizionali** servono proprio a questo. La forma più semplice di istruzione condizionale è l'istruzione `if` ("se" in inglese):

```
if x > 0:
    print('x è positivo')
```

L'espressione booleana dopo l'istruzione `if` è chiamata **condizione**. Se risulta vera, viene eseguita l'istruzione indentata che segue sulla riga successiva. Altrimenti, non succede nulla.

L'istruzione `if` ha la stessa struttura che abbiamo già visto nel caso delle definizioni di funzione: un'intestazione seguita da un corpo indentato. Le istruzioni come questa vengono chiamate **istruzioni composte**.

Non c'è limite al numero di istruzioni che possono essere scritte nel corpo, ma deve sempre essercene almeno una. Talvolta può servire che il corpo sia privo di istruzioni (di solito quando c'è del codice ancora da scrivere); in questo caso potete usare l'istruzione `pass`, che serve solo da segnaposto temporaneo e nulla più:

```
if x < 0:
    pass          # scrivere cosa fare con i valori negativi!
```

## 5.5 Esecuzione alternativa

Una seconda forma di istruzione `if` è l'**esecuzione alternativa**, dove esistono due possibili azioni, e il valore della condizione determina quale delle due azioni debba essere eseguita e quale no. La sintassi è:

```
if x % 2 == 0:
    print('x è pari')
else:
    print('x è dispari')
```

Se il resto della divisione di `x` per 2 è zero, significa che `x` è un numero pari, e il programma mostra il messaggio appropriato. Altrimenti (`else`), se la condizione è falsa, viene eseguito il secondo blocco di istruzioni. Dato che la condizione deve essere necessariamente o vera o falsa, sarà sempre eseguita una sola delle due alternative. Queste sono dette **ramificazioni**, perché rappresentano dei bivi nel flusso di esecuzione del programma.

## 5.6 Condizioni in serie

A volte è necessario considerare più di due possibili sviluppi, e occorre che nel programma ci siano più di due ramificazioni. Un modo per esprimere questo tipo di calcolo sono le **condizioni in serie**:

```
if x < y:
    print('x è minore di y')
elif x > y:
    print('x è maggiore di y')
else:
    print('x e y sono uguali')
```

`elif` è l'abbreviazione di *else if*, che in inglese significa "altrimenti se". Anche stavolta verrà eseguito solo uno dei tre rami, a seconda dell'esito del confronto tra `x` e `y`. Non c'è alcun limite al numero di istruzioni `elif`. Se esiste una clausola `else`, deve essere scritta per ultima, ma non è obbligatoria; il ramo corrispondente viene eseguito solo quando tutte le condizioni precedenti sono false.

```
if scelta == 'a':
    disegna_a()
elif scelta == 'b':
    disegna_b()
elif scelta == 'c':
    disegna_c()
```

Le condizioni vengono controllate nell'ordine dall'alto al basso: se la prima è falsa, viene controllata la seconda e così via. Non appena una condizione risulta vera, viene eseguito il ramo corrispondente e l'istruzione termina. Anche se risultassero vere altre condizioni successive, viene eseguita sempre e soltanto la prima che risulta vera.

## 5.7 Condizioni nidificate

Si può anche inserire un'istruzione condizionale nel corpo di un'altra istruzione condizionale. Possiamo dunque scrivere l'esempio del paragrafo precedente anche in questo modo:

```
if x == y:
    print('x e y sono uguali')
else:
    if x < y:
        print('x è minore di y')
    else:
        print('x è maggiore di y')
```

La condizione più esterna contiene due rami: il primo contiene un'istruzione semplice, il secondo un'altra istruzione `if` che a sua volta ha due ramificazioni. Entrambi i rami del secondo `if` sono istruzioni di stampa, ma potrebbero anche contenere a loro volta ulteriori istruzioni condizionali.

Anche se l'indentazione delle istruzioni aiuta ad evidenziare la struttura, le **condizioni nidificate** diventano rapidamente difficili da leggere, quindi è meglio usarle con moderazione.

Gli operatori logici permettono spesso di semplificare le istruzioni condizionali nidificate. Il codice seguente può essere riscritto usando un'unica condizione:

```
if 0 < x:
    if x < 10:
        print('x è un numero positivo a una cifra.')
```

Infatti, dato che l'istruzione di stampa è eseguita solo se si verificano entrambe le condizioni, possiamo ottenere lo stesso risultato usando l'operatore `and`:

```
if 0 < x and x < 10:
    print('x è un numero positivo a una cifra.')
```

Per una condizione di questo tipo, Python consente anche un'opzione sintattica più concisa:

```
if 0 < x < 10:
    print('x è un numero positivo a una cifra.')
```