La funzione si definisce con def e termina con end.
Il while, if ed il for terminano con un end.
ARGV è un array che contiene gli argomenti passati da riga di comando.
.to_i converte una stringa di caratteri in un intero.
Ruby non fa distinzione tra dichiarazioni ed espressioni.
“ ” permette di inserire i caratteri preceduti da backslash e la valutazione di espressioni contenute in essa usando #{} es. “abcd #{5*3} efg” mostrerà “abcd 15 efg”
Una stringa si concatena con il +
Gli indici negativi indicano la distanza dalla fine di una stringa, invece che dall'inizio.
es. herb= “parsley”
“parsley”
herb[0,1]
“p”
herb[-2,2]
“ey”
herb[0..3]
“pars”
herb[-5..-2]
“rsale”
=~ è l'operatore di confronto delle espressioni regolari.
Has è un array associativo dove gli elementi sono identificati attraverso le chiavi che possono avere qualsiasi tipo di lavoro. es.
H={1 =>2, “2”=>”4”}
h[1]
2
h[“2”]
“4”
h[5]= 10 Aggiunge un valore
h
{5=>10, 1=>2, “2” =>”4”}
h.delete 1 Cancella un valore
2
h[1]
nil Non esiste il valore poichè è stato cancellato, non scalano i valori.
h
{5=>10, “2”=>”4”}
rand restituisce un numero casuale
STDIN.gets memorizza quello che è stato digitato da tastiera.
.chop! rimuove l'ultimo carattere. es.
testo.chop! ritornerà test
s1=”forth”
s1.chop!
“fort”
s2=s1.chop Il chop senza il ! fa una copia in s2
“for”

each.byte è l'iteratore per ogni carattere della stringa
each_line

Per esempio la fabbrica è una classe ed il prodotto finale è un oggetto. Per es. i bottoni e quadranti sono i metodi degli oggetti.
Una classe è una collezione di funzioni speciali chiamati metodi e variabili speciali chiamate proprietà. Una classe può essere dichiarata mediante la parola class. Le classi sono i modelli a partire dai quali vengono creati gli oggetti.
Un oggetto è la versione operativa della funzionalità definita nella classe. Un oggetto viene istanziato con l'istruzione new specificata insieme al nome della classe di cui esso dovrà essere un membro.

Metodi o funzioni

Una volta istanziato un oggetto, diventa possibile accedere a tutte le sue proprietà e a tutti i suoi metodi. es.
"abcdefg".lenght (il metodo dell'oggetto abcdefg che dice quanto è lunga la frase)
7
In ruby c'è la variabile self, essa che si riferisce a qualunque oggetto stia chiamando un metodo. es.
Self.nome_metodo(args.....)

Classi

Una categoria di oggetti come il cane è una classe. Le istanze sono specifici oggetti che appartengono a questa classe. Per creare un oggetto in ruby prima bisogna definire le caratteristica della classe e quindi creare una istanza. es. Definiamo la classe Cane:

```
class Cane
  def abbaia    #è un metodo della classe, è un comportamento di un oggetto di quella classe.
    print "Bau Bau\n"
 end
end
```

Creiamo una istanza della classe Cane chiamandola fido.

```
fido=Cane.new
```

Il metodo new di ogni classe crea una nuova istanza. Adesso diciamo cosa far fare al cane.

```
fido.abbaia  #abbaia è il metodo def
```

Bau Bau
Se scrivessimo Cane.abbaia avremmo un errore se invece scrivessimo (Cane.new).abbaia avremmo
Bau Bau
In quest'ultimo esempio però scompare, lo invochiamo solo una volta.

Ereditarietà

I gatti sono mammiferi ed i mammiferi sono animali. Le classi minori ereditano le caratteristiche dalle classi maggiori a cui appartengono. Se i mammiferi respirano, allora respirano anche i gatti.

```
class Mammifero
  def respiro
      print "inspira ed espira\n"
  end
end
class Gatto<Mammifero
  def miagolak
      print "Miao\n"
```

```
  end
end
```
Gatto è una sottoclasse di Mammifero, la classe maggiore è detta superclasse e la minore è detta sottoclasse. Aggiungiamo il metodo miagola così il gatto può sia respirare che miagolare.
Creiamo una istanza della classe Gatto chiamandola tama.
```
tama = Gatto.new
tama.respira
inspira ed espira
tama.miagola
Miao
```
In alcune situazioni determinati comportamenti di una superclasse non devono essere ereditati da una sottoclasse.

```
class Uccello
  def pulisci
      print “Sto pulendo le mie penne\n”
  end
  def vola
       print “Sto volando.”
  end
end
class Pinguino<Uccello
  def vola
     fail “Spiacente. Preferisco nuotare.”
  end
end
```
Anziché definire interamente ogni caratteristica di ogni nuova classe, possiamo aggiungere o ridefinire le caratteristiche differenti tra ogni sottoclasse e la sua superclasse.

Ridefinizione dei metodi

In una sottoclasse, possiamo cambiare il comportamento delle istanze attarverso la ridefinizione dei metodi della superclasse.
```
class Umano
  def identificazione
     print “Sono una persona.\n”
  end
  def biglietto_treno(anni)
     if anni < 12
       print “Biglietto ridotto.\n”;
     else
       print “Biglietto normale.\n”;
     end
  end
end
Umano.new.identificazione
Sono una persona
class Studente1<Umano
  def identificazione
    print “Sono uno studente.\n”
  end
```

```
end
Studente1.new.identificazione
Sono uno studente
```
Supponiamo di voler migliorare il metodo identificazione della superclasse invece di sostituirlo. Per farlo possiamo usare super.
```
class Studente2<Umano
  def identificazione
      super
       print “Sono anche uno studente.\n”
  end
end
Studente2.new.idenficazione
Sono una persona.
Sono anche uno studente.
```
Super permette di passare degli argomenti al metodo originale.

```
Disonesto<Umano
  def biglietto_treno(anni)
    super(11)
  end
end
Disonesto.new.biglietto_treno(25)
Biglietto ridotto  #perchè minore di 12
class Onesto<Umano
  def biglietto_treno(anni)
    super(anni)  #passa l'argomento dato da noi
  end
end
Onesto.new.biglietto_treno(25)
Biglietto normale.  #perchè maggiore di 12
```