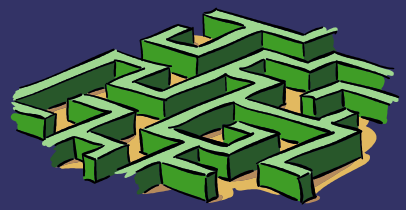
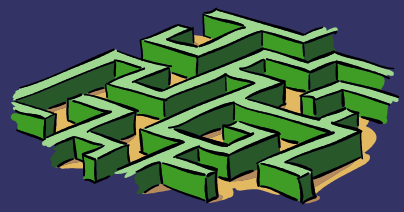
Perl, OOP e BioPerl

- Programma di oggi:
  - BioPerl e programmazione a oggetti:
  - (a che vi servono?)
- La programmazione ad oggetti in Perl
  - Funzioni e moduli
  - Oggetti
- Il Tutorial BioPerl
  - Esempi d'uso



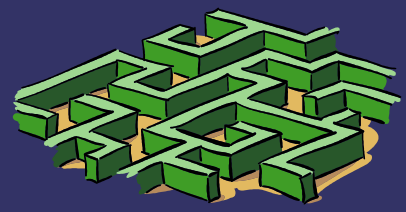
A che vi servono BioPerl e la programmazione a oggetti?

- BioPerl vi permette di:
  - Manipolare diversi tipi di sequenze: DNA, RNA, Proteine ...
  - Leggerle e scriverle da: file, testo, database locali, database remoti
  - Usare diversi formati di memorizzazione: fasta, swiss, embl, genbank, gcg, ...
  - Usare e integrare diverse applicazioni bioinformatiche: BLAST, EMBOSS, PAML, PDB, PISE, ...
- BioPerl è una libreria formata da **oggetti**
  - È facile imparare ad **usare** gli oggetti
  - (abbastanza facile crearne nuovi tipi)



Programmazione ad oggetti (OOP)

- Cosa è un oggetto?
  - È l'unione di una **struttura** dati con le **azioni** che la manipolano
  - Un programma ad oggetti non si basa su una sequenza di istruzioni (stile procedurale) ma si basa sulla interazione tra gli oggetti
- Esempio:
  - Figura geometrica in un editor grafico
    - **Proprietà**: Posizione, Colore, Dimensioni
    - **Azioni**: Disegna, Ridimensiona, Stampa, Sposta, Cambia colore, Calcola area, Vedi se interseca un altro elemento grafico



La vita (uso) di un oggetto

- Per usarlo dobbiamo **conoscerne la definizione** (ad esempio scritta da altri):

```
use Rettangolo;
```

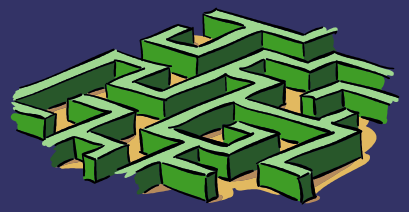
- **Nascita**: creiamo uno specifico rettangolo

```
my $rettangolo = Rettangolo ->new(10,20);
```

- **Vita**: lo manipoliamo

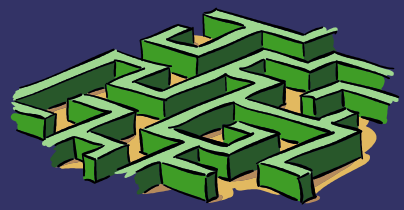
```
$rettangolo -> muovi(100,200);
$rettangolo -> colore('rosso');
my $area = $rettangolo -> area();
$rettangolo -> mostra();
```

- **Morte**: lo distruggiamo
  - Perl ha il garbage-collector automatico



Come si definisce un oggetto

- **Classe:** descrizione della **Struttura** dati e delle sue proprietà
  - Serve a creare molteplici **esemplari** di quel tipo di struttura (**Istanze**)
  - Contiene i **Metodi**: ovvero le **Azioni** con cui manipolare gli oggetti di questo tipo
- Esempio:
  - Rettangolo
    - **Proprietà**: Posizione, Colore, Dimensioni
    - **Azioni**: Disegna, Ridimensiona, Stampa, Sposta, Cambia colore, Calcola area, Vedi se interseca un altro elemento grafico



L'OOP semplifica lo sviluppo

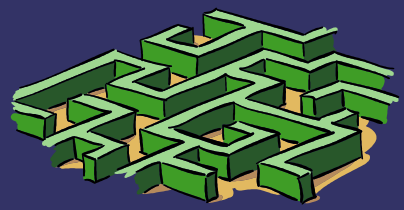
- **Ereditarietà:** una nuova classe “estende” le funzionalità di una classe già presente
  - → Permette il riuso efficiente del codice

  Esempio:

  Un quadrato è un rettangolo con i lati uguali
- **Incapsulamento:** nascondere i dettagli interni dell'implementazione
  - → rende facile fare modifiche senza dover toccare il resto dell'applicazione

  Esempio:

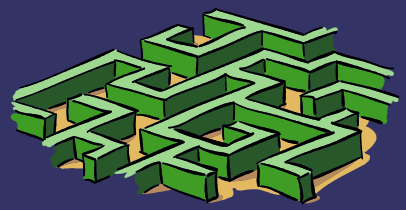
  Usare coordinate cartesiane o polari per le posizioni



Richiami: Funzioni in Perl

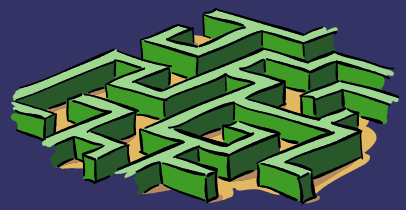
- Implementano una azione, un calcolo
- Ricevono una lista di parametri di lunghezza *variabile*
- Possono fornire *più di un risultato*
- Usano variabili *locali*, non accessibili all'esterno

```
sub calcolaQuadrato {
my ($x) = @_ ;
my $q = $x * $x;
return $q;
}
```



Richiami: Moduli in Perl

- In un modulo si possono definire:
  - Funzioni
  - Variabili globali
- Le funzioni e variabili definiti nel modulo sono accessibili solo se lo si include nel proprio programma
- I moduli permettono sia lo stile di programmazione procedurale che lo stile object-oriented

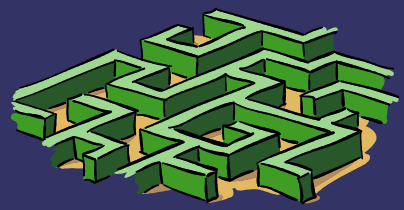


Come è fatta la OOP in Perl

- A qualsiasi struttura dati è possibile associare un Modulo
  - In questo modo si associano alla struttura dati le azioni (metodi) che la debbono/possono manipolare
- Ogni oggetto “sa” quali sono le sue azioni:
  - quando si scrive:

    $oggetto -> metodo (*argomenti*);
  - in realtà si esegue l'azione:

    'Modulo::metodo'($oggetto, *argomenti*)
- Un caso particolare: il metodo '**new**' per creare una nuova istanza
  - Esempio: **my $uovo = Uovo -> new('2 Feb 04');**
  - Si possono usare altri nomi oltre a 'new'

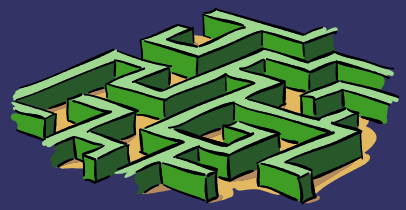


... segue OOP in Perl

- Ereditarietà: Si usa la variabile **@ISA** per elencare le classi che il modulo estende

```
package Quadrato;
@ISA = qw( Rettangolo );
```

  - Quando si chiama un metodo esso viene cercato:
    - Nel package dell'oggetto
    - Nei package elencati nella variabile @ISA ... e così via
- Incapsulamento: è più uno stile di programmazione che una caratteristica del linguaggio
  - Per convenzione si usano **solo i metodi** dell'oggetto
    - È meglio non usare i nomi delle sue proprietà direttamente
    - I metodi 'privati' dell'oggetto iniziano per '_' (underscore) e non vanno usati



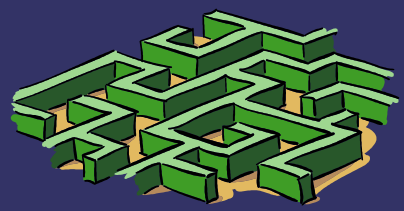
Definiamo un primo oggetto

```
package Rettangolo;
sub new {
    my ($classe, $w, $h) = @_ ;
    my $self = {
        x => 0, y=> 0,
        largh => $w, alt => $h,
        colore => 'bianco' };
    return bless $self;
}

sub area {
    my ($self) = @_ ;
    return    $self->{larg} *
                $self->{alt};
}
```

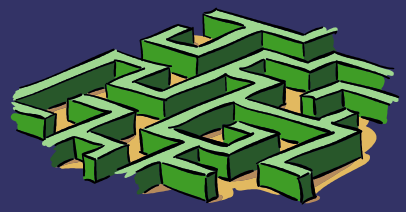
```
sub mostra {
    my ($classe) = @_ ;
    print “\nPosizione: ”,
        $self->{x}, $self->{y};
    print “\nColore: ”,
        $self->{colore};
    print “\nDimensioni: ”,
        $self->{largh},
        $self->{alt};
    print “\nArea: ”, $self->area;
}

sub muovi {
    my ($self, $x, $y) = @_ ;
    $self->{x} = $x;
    $self->{y} = $y;
}
```

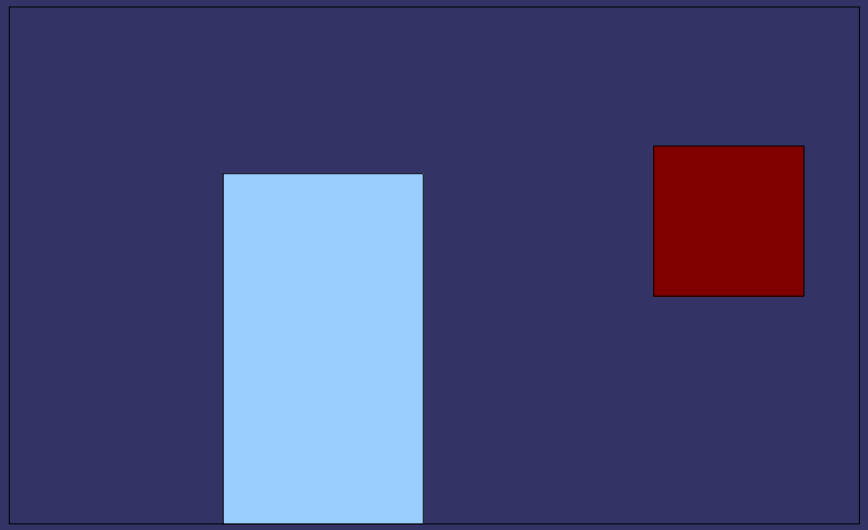


Un quadrato è un rettangolo con lati uguali

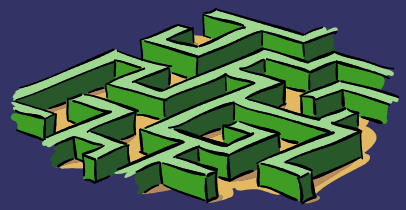
```
package Quadrato;
use Rettangolo;
@ISA = qw( Rettangolo );
sub new {
    my ($classe, $lato) = @_ ;
    my $self = Rettangolo->new($lato,$lato);
    return bless $self;
}
```



Usiamo gli oggetti

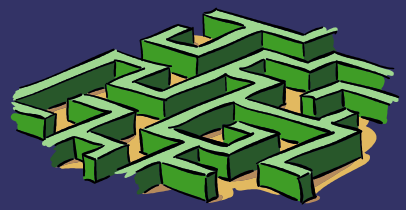


```
use Quadrato;
use Rettangolo;
my $porta = Rettangolo -> new(100,200);
$porta -> {colore} = 'azzurro';
$porta -> muovi(100, 0);
my $quadro = Quadrato -> new(50);
$quadro -> {colore} = 'red';
$quadro -> muovi(300, 160);
$porta -> mostra();
$quadro -> mostra();
```



Un oggetto 'bio'

- Sequenza di basi di un frammento di DNA
  - Proprietà:
    - Sequenza di basi indicate dalle lettere AGCT
    - Posizione corrente
    - ...
  - Azioni:
    - Crea una sequenza leggendo i dati da un file in formato fasta
    - Genera la sequenza complementare
    - Cerca la posizione del primo codone valido
    - Salva la sequenza in formato genbank
    - Calcola il peso molecolare della sequenza



Alcune classi di BioPerl

- Bio::Seq
  - Sequenza di DNA, RNA, Proteina ...
- Bio::SeqIO
  - Convertitore di formati: fasta, genbank, ...
- Bio::Annotation
  - Gestisce le informazioni allegate alle sequenze
- Bio::Location
  - Posizione in una sequenza
- Bio::Structure
  - Struttura 3d di proteine
- Bio::DB
  - Accesso a database locali e remoti
- ...