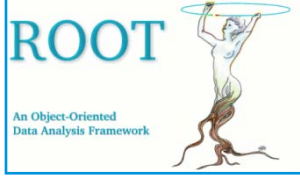
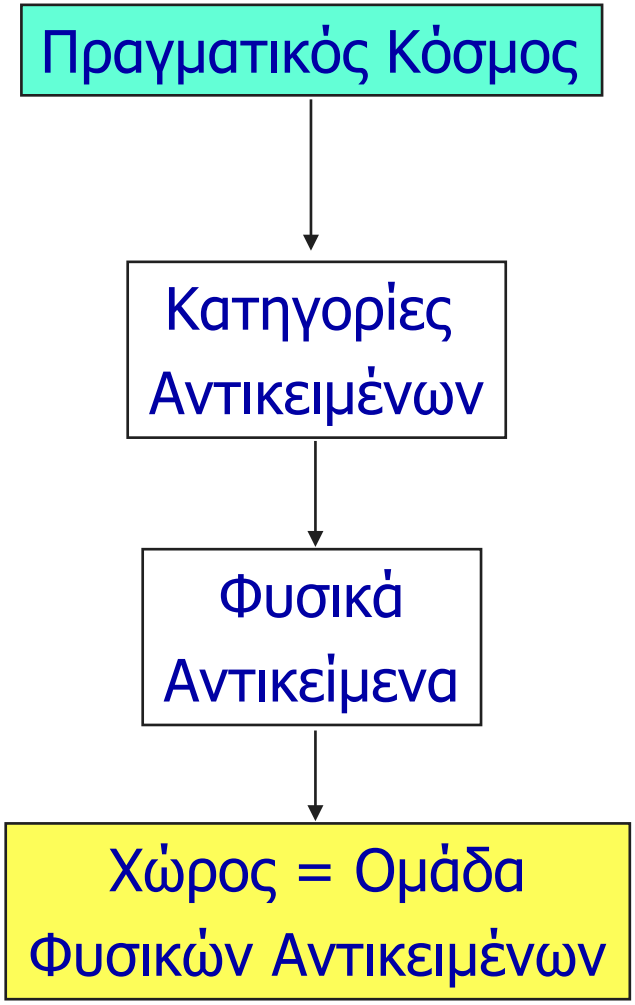
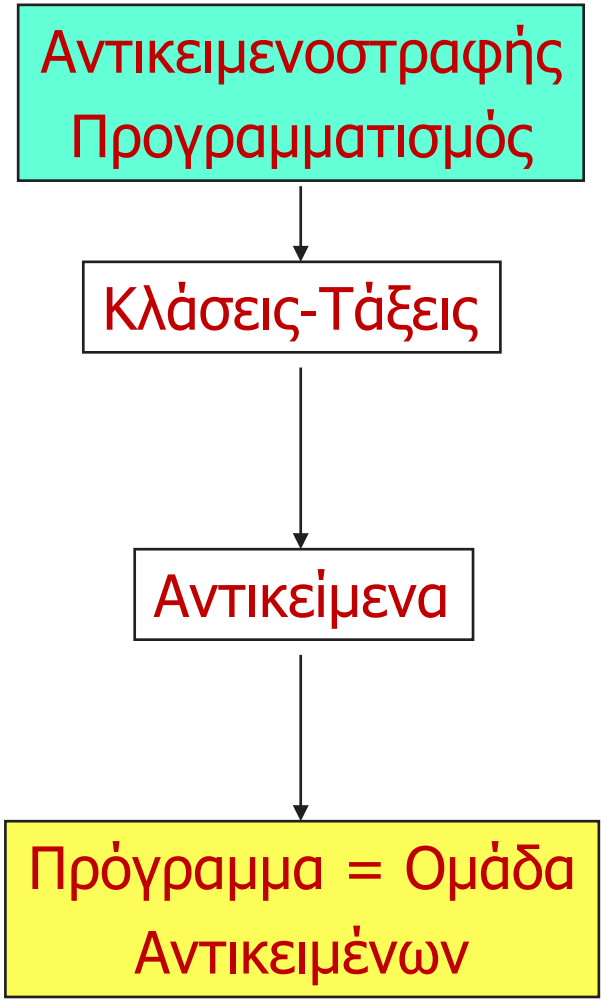
Κλάσεις

- Αντικειμενοστραφής Προγραμματισμός
- Κλάσεις-Αντικείμενα
- Ένα παράδειγμα
- Συναρτήσεις κατασκευής (Constructors)
- Συνάρτηση καταστροφής (Destructor)
- Συναρτήσεις πρόσβασης (Access Functions)
- Συνάρτηση κατασκευής αντίγραφου
- Δείκτες προς αντικείμενα.
- Στατικά μέλη δεδομένων
- Στατικές συναρτήσεις-μέλη

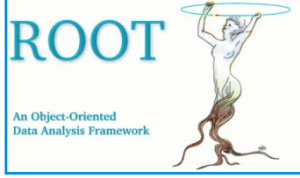
Αντικειμενοστραφής Προγραμματισμός



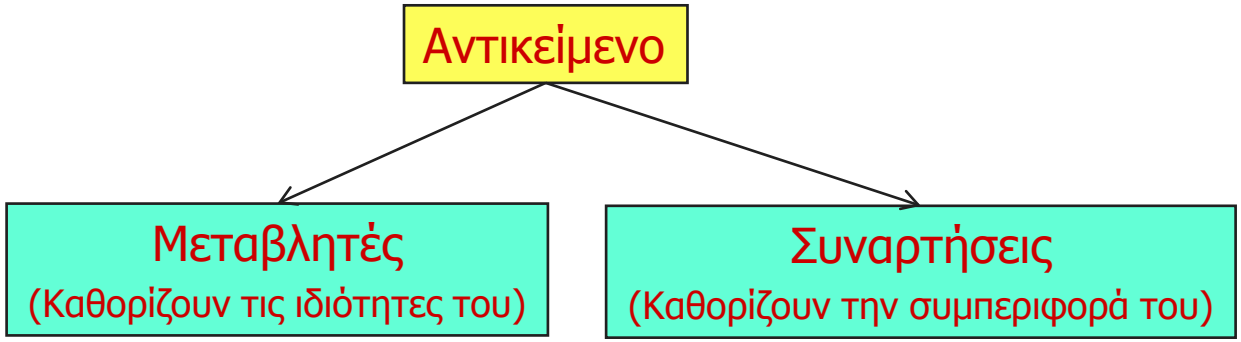




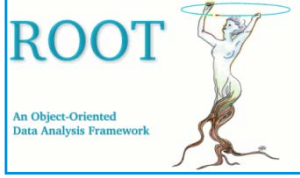
Κλάσεις-Αντικείμενα



- Η κλάση (class) είναι ένας παράγωγος τύπος του οποίου τα στοιχεία δύναται να είναι διαφορετικών τύπων. Επιπλέον, ορισμένα στοιχεία μιας κλάσης μπορεί να είναι συναρτήσεις και τελεστές.
- Ως αντικείμενο μπορεί να θεωρηθεί οποιαδήποτε “οντότητα ” η οποία καταλαμβάνει περιοχή αποθήκευσης στη μνήμη. Παρ’ όλα αυτά η λέξη χρησιμοποιείται συνήθως για να περιγράψει μεταβλητές των οποίων ο τύπος είναι μια κλάση.
- Θεωρούμε το αντικείμενο ως μια αυτάρκη οντότητα που αποθηκεύει τα δικά του δεδομένα και τις δικές του συναρτήσεις. Η λειτουργικότητα ενός αντικειμένου του δίνει ζωή με τη έννοια ότι “γνωρίζει” πώς να κάνει μόνο του διάφορα πράγματα.
- Οι κλάσεις χρησιμοποιούνται ως *πρότυπα* για την δημιουργία των αντικειμένων.



Κλάσεις-Αντικείμενα



Γενικό διάγραμμα κλάσης

```
class Ratio
{
   public:
      Ratio(int,int);
      ~Ratio();
      int numerator();
      int denominator();
      double convert();
      void invert();
      void print();
   private :
      int num, den;
}
```

Όνομα Κλάσης

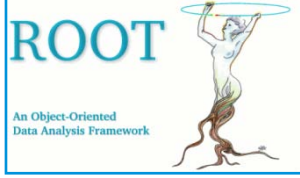
Συναρτήσεις-Μέλη (member functions)

Μέθοδοι (methods)
Υπηρεσίες (services)

Μέλη-Δεδομένων (member data)

Στην παραπάνω κλάση όλες οι συναρτήσεις-μέλη έχουν ορισθεί ως δημόσιες (public) και όλα τα μέλη δεδομένων ως ιδιωτικά (private).

Κλάσεις-Αντικείμενα



Γενικό διάγραμμα κλάσης

```
class ΟνομαΚλάσης
{
        public:
```

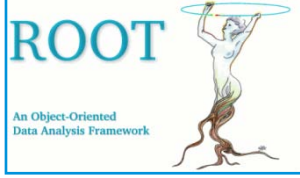
- Συναρτήσεις κατασκευής (Constructors)
- Συνάρτηση καταστροφής (Destructor)
- Συναρτήσεις πρόσβασης (Access Functions)
- Συναρτήσεις:
  - Κατασκευής Αντιγράφου
  - Εκτύπωσης
  - Που καθορίζουν την συμπεριφορά των αντικειμένων

```
        private:
```

- Μέλη δεδομένων (member data)

```
}
```

Παράδειγμα – Κλάση Ρητών Αριθμών




```
// Example Class Ratio
// Paradeigma klasis ritwn arithmwn

#include <iostream>
using namespace std;

class Ratio
{ public:
    Ratio(int, int);      //Constructor
    ~Ratio();             //Destructor
    int numerator();      //Access function
    int denominator();    //Access function
    double convert();
    void invert();
    void print();
  private:
    int num, den;
};

Ratio::Ratio(int numerator, int denominator)
{ num = numerator;
  den = denominator;
}

Ratio::~Ratio()
{   } //Empty

int Ratio::numerator()
{ return num;}

int Ratio::denominator()
{ return den; }
```

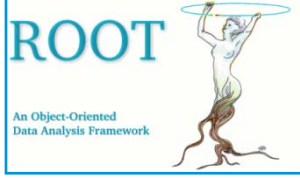
Δήλωση Κλάσης Ratio

Συνάρτηση κατασκευής (Constructor)

Συνάρτηση καταστροφής (destructor)

Συναρτήσεις πρόσβασης (Access Functions)

Παράδειγμα – Κλάση Ρητών Αριθμών



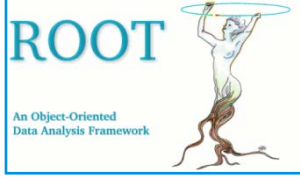

```
double Ratio::convert()
{ return double(num)/den;
}

void Ratio::invert()
{ int temp = num;
  num = den;
  den = temp;
}

void Ratio::print()
{ cout << num << '/' << den;
}

////////////////////////////////////////////////////////////////

int main()
{ Ratio x(22,7);
  cout << "x = ";
  x.print();
  cout << " = " << x.convert() << endl;
  x.invert();
  cout << "1/x = ";  x.print();
  cout << endl;
}
```

Διάφορες Συναρτήσεις

Συνάρτηση εκτύπωσης

```
[panos@pc-247 Cpp]$ c++ Ratio.cpp
[panos@pc-247 Cpp]$ a.out
x = 22/7 = 3.14286
1/x = 7/22
[panos@pc-247 Cpp]$
```

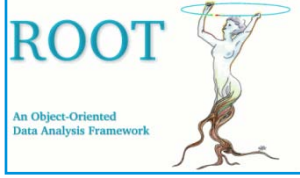
Παράδειγμα – Κλάση Ρητών Αριθμών



Μια αυτάρκης υλοποίηση της κλάσης Ratio

```
// Example Class Ratio
// Paradeigma klasis ritwn arithmwn

#include <iostream>
using namespace std;

class Ratio
{ public:
    Ratio(int n, int d) { num=n; den=d; }      //Constructor
    ~Ratio() {  }                              //Destructor
    int numerator()   { return num;}           //Access function
    int denominator() { return den;}           //Access function
    double convert() { return double(num)/den; }
    void invert() { int temp = num; num = den; den = temp; }
    void print() { cout << num << '/' << den;}
  private:
    int num, den;
};
```

Συναρτήσεις κατασκευής (Constructors)



- Η συνάρτηση κατασκευής (constructor) είναι μια συνάρτηση μέλος η οποία καλείται αυτόματα κατά τη δήλωση ενός αντικειμένου. Πρέπει να έχει το ίδιο όνομα με την κλάση και δηλώνεται χωρίς επιστρεφόμενο τύπο.
- Στο παράδειγμά μας:

```
class Ratio
{ public:
    Ratio(int, int);
    ~Ratio();
    int numerator();
    int denominator();
    double convert();
    void invert();
    void print();
  private:
    int num, den;
};
```

```
Ratio::Ratio(int numerator, int denominator)
{ num = numerator;
  den = denominator;
}
```

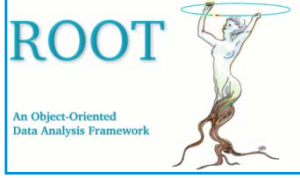
- Η κατασκευή ενός αντικειμένου γίνεται απλά ως:

  Ratio x(34,78), y(12,78);

- Μία κλάση μπορεί να έχει περισσότερες από μία συναρτήσεις κατασκευής.

```
class Ratio
{ public:
    Ratio() { num=0; den=1; }                  //Constructor
    Ratio(int n) { num=n; den=1; }             //Constructor
    Ratio(int n, int d) { num=n; den=d; }      //Constructor
    ~Ratio() {  }                              //Destructor
```

Συναρτήσεις κατασκευής (Constructors)



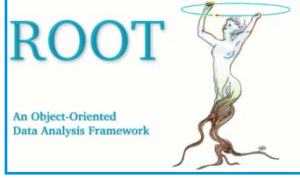
- Η C++ περιλαμβάνει έναν ειδικό συντακτικό μηχανισμό για απόδοση αρχικών τιμών μέσω των συναρτήσεων κατασκευής. Ο μηχανισμός αυτός είναι η λίστα απόδοσης αρχικών τιμών (initialization list). Πχ: περιλαμβάνει

  Ratio(int n, int d) : num(n), den(d) { }

- Ποιο συγκεκριμένα στο παράδειγμά μας μπορούν να γίνουν οι ακόλουθες υλοποιήσεις:

```
class Ratio
{ public:
    Ratio() : num(0), den(1){ }                  //Constructor
    Ratio(int n) : num(n), den(1) { }            //Constructor
    Ratio(int n, int d) : num(n), den(d) { }     //Constructor
  private:
    int num, den;
};



class Ratio
{ public:
    Ratio(int n=0, int d=1) : num(n), den(d) { } //Constructor
  private:
    int num, den;
};
```

Συνάρτηση καταστροφής (Destructor)

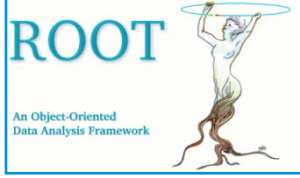


- Όταν δημιουργείται ένα αντικείμενο, καλείται αυτόματα η συνάρτηση κατασκευής. Αντίστοιχα όταν ένα αντικείμενο φτάνει στο τέλος της ζωής του, καλείται αυτόματα η ειδική συνάρτηση-μέλος που ονομάζεται συνάρτηση καταστροφής (destructor).
- Κάθε κλάση έχει μία μόνο συνάρτηση καταστροφής. Ένα η συνάρτηση καταστροφής δεν ορισθεί ρητά, δημιουργείται αυτόματα από το σύστημα (όπως και στην περίπτωση της συνάρτησης κατασκευής).
- Στο παράδειγμά μας ορίζουμε μια κενή συνάρτηση καταστροφής:

```
class Ratio
{ public:
    Ratio(int, int);
    ~Ratio();
    int numerator();
    int denominator();
    double convert();
    void invert();
    void print();
  private:
    int num, den;
};
```

```
Ratio::~Ratio()
{   }  //Empty
```

Συναρτήσεις πρόσβασης (Access Functions)



- Τα μέλη δεδομένων μιας κλάσης συνήθως δηλώνονται ως ιδιωτικά (private). Ο λόγος είναι η προστασία τους (information hiding). Για να έχουμε πρόσβαση στα δεδομένα απαιτούνται οι δημόσιες (public) συναρτήσεις πρόσβασης (access functions) .
- Στο παράδειγμά μας έχουμε δύο συναρτήσεις πρόσβασης:

```
class Ratio
{ public:
    Ratio(int, int);
    ~Ratio();
    int numerator();
    int denominator();
    double convert();
    void invert();
    void print();
  private:
    int num, den;
};
```

```
int Ratio::numerator()
{ return num;}

int Ratio::denominator()
{ return den; }
```

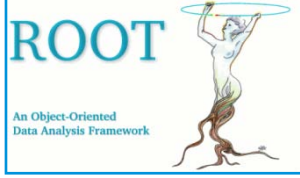
- Έτσι εάν:

```
Ratio x(23,67);
x.numerator();
x.denominator();
```

→ Επιστρέφει τον αριθμητή: 23

→ Επιστρέφει τον παρονομαστή: 67

Συνάρτηση κατασκευής αντίγραφου



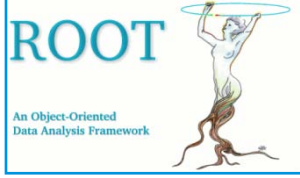
- Κάθε κλάση εκτός της συνάρτησης κατασκευής έχει και την συνάρτηση κατασκευής αντιγράφου (copy constructor).

- Όταν καλείται η συνάρτηση κατασκευής αντιγράφου, αντιγράφει την πλήρη κατάσταση ενός υπάρχοντος αντικειμένου σε ένα νέο αντικείμενο της ίδιας κλάσης.
- Αν ο ορισμός της κλάσης δεν περιλαμβάνει ρητά μια συνάρτηση κατασκευής αντιγράφου, τότε το σύστημα εξ ορισμού δημιουργεί αυτόματα μία.
- Η συνάρτηση δέχεται ως όρισμα το αντικείμενο που πρόκειται να αντιγράψει, το οποίο μεταβιβάζεται κατά σταθερή αναφορά.

```
// Example Copy Constructor

#include <iostream>
using namespace std;

class Ratio
{ public:
    Ratio(int n=0, int d=1) : num(n), den(d) { }
    Ratio(const Ratio& r) : num(r.num), den(r.den) { }
    void print() { cout << num << '/' << den; }
  private:
    int num, den;
};

int main()
{ Ratio x(100,360);
  Ratio y(x);
  cout << "x = ";
  x.print();
  cout << endl;
  cout << "y = ";
  y.print();
  cout << endl;
}
```

```
[panos@pc-247 Cpp]$ c++ syn_antigrafou.cpp
[panos@pc-247 Cpp]$ a.out
x = 100/360
y = 100/360
[panos@pc-247 Cpp]$
```

Δείκτες προς αντικείμενα



- Σε πολλές εφαρμογές είναι πλεονεκτική η χρήση δεικτών προς αντικείμενα.
- Οι δύο συμβολισμοί

  (*px).print()

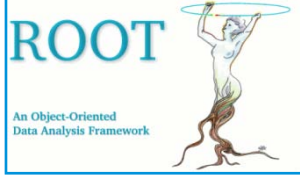
  px->print()

  έχουν την ίδια σημασία.
- Όταν χειρίζεστε δείκτες είναι προτιμητέο το σύμβολο του “βέλους” “->”.

```
using namespace std;

class Ratio
{ public:
    Ratio(int n=0, int d=1) : num(n), den(d) {  }
    void print() { cout << num << '/' << den; }
  private:
    int num, den;
};

int main()
{ Ratio x(100,360);
  Ratio* px = &x;

  cout << "x = " ;
  x.print();
  cout << endl;

  cout << "(*px).print() = ";
  (*px).print();
  cout << endl;

  cout << "  px->print() = ";
  px->print();
  cout << endl;
}
```

```
[panos@pc-247 Cpp]$ c++ class_deiktes.cpp
[panos@pc-247 Cpp]$ a.out
x = 100/360
(*px).print() = 100/360
  px->print() = 100/360
[panos@pc-247 Cpp]$
```

Στατικά μέλη δεδομένων

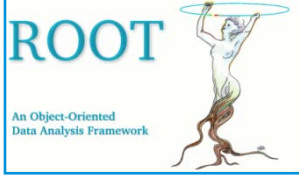


- Μερικές φορές, μια τιμή ενός μέλους δεδομένων ισχύει για όλα τα μέλη της κλάσης. Για να αποφεύγουμε να αποθηκεύουμε την ίδια τιμή για κάθε αντικείμενο της κλάσης χρησιμοποιούμε τη λέξη κλειδί static στην δήλωση της μεταβλητής.
- Κάθε static μεταβλητή πρέπει να ορισθεί καθολικά.
- Ένα στατικό μέλος δεδομένων μοιάζει με μια καθολική μεταβλητή: υπάρχει μόνον ένα αντίγραφο της μεταβλητής ανεξάρτητα με τον αριθμό των αντικειμένων.
- Με απλούστερα λόγια τα στατικά μέλη δεδομένων μπορούμε να θεωρήσουμε πως “ανήκουν” κατ’ ευθείαν στις κλάσεις και όχι στα αντικείμενα αυτής. Γιαυτό και η κλήση τους μπορεί να γίνει απλά και με τον τελεστή εμβέλειας “::”.

```
// Example static variable

#include <iostream>
using namespace std;

class Ratio
{ public:
    Ratio(int n=0, int d=1) : num(n), den(d) { }
    void print() { cout << num << '/' << den; }
    static int count;
  private:
    int num, den;
};

int Ratio::count = 0;   //Katholikos orismos

int main()
{ Ratio x, y;

  x.count=10;

  cout << "y.count = " << y.count << endl;

  cout << "Ratio::count = " << Ratio::count << endl;

}
```

```
[panos@pc-247 Cpp]$ c++ static.cpp
[panos@pc-247 Cpp]$ a.out
y.count = 10
Ratio::count = 10
[panos@pc-247 Cpp]$
```

Στατικές συναρτήσεις-μέλη



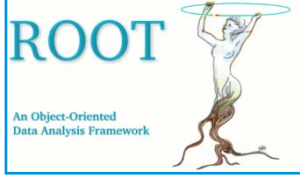
- Κατ' αντιστοιχία των στατικών δεδομένων έχουμε και τις στατικές συναρτήσεις μέλη . Η δράση των στατικών συναρτήσεων είναι ανεξάρτητη από τα πραγματικά αντικείμενα της κλάσης.

- Όπως φαίνεται στο παράδειγμα η δήλωση της στατικής συνάρτησης number() την καθιστά ανεξάρτητη από τα στιγμιότυπα της κλάσης.
- Η συνάρτηση καλείται απλά ως μέλος της κλάσης με την χρήση του τελεστή εμβέλειας "::".
- Οι στατικές συναρτήσεις μέλη μπορούν να προσπελάσουν μόνο στατικά δεδομένα της κλάση τους.

```
// Example static variable

#include <iostream>
using namespace std;

class Ratio
{ public:
    Ratio(int n=0, int d=1) : num(n), den(d) { }
    void print() { cout << num << '/' << den; }
    static int number() { return count; }
  private:
    int num, den;
    static int count;
};

int Ratio::count = 10;   //Katholikos orismos

int main()
{
  cout << "count = " << Ratio::number() << endl;
}
```

```
[panos@pc-247 Cpp]$ c++ static_func.cpp
[panos@pc-247 Cpp]$ a.out
count = 10
[panos@pc-247 Cpp]$
```

Παράδειγμα



Υλοποιήστε την κλάση Point3D για σημεία σε τρεις διαστάσεις (x,y,z). Συμπεριλάβετε μια προεπιλεγμένη συνάρτηση κατασκευής, μια συνάρτηση κατασκευής αντιγράφου, τη συνάρτηση negate() για τη μετατροπή του σημείου στο αρνητικό του, τη συνάρτηση norm() η οποία επιστρέφει την απόσταση του σημείο από την αρχή (0,0,0) των αξόνων και μια συνάρτηση print().

```
// Example point3D

#include <iostream>
#include<cmath>
using namespace std;

class Point3D
{ public:
    Point3D(float a=0, float b=0, float c=0) : x(a), y(b), z(c) {  }
    Point3D(const Point3D& p) :  x(p.x), y(p.y), z(p.z) {  }
    void negate() { x *=-1;  y *=-1; z *=-1; }
    double norm() { return sqrt(x*x + y*y + z*z); }
    void print() { cout << "(" << x << "," << y << "," << z << ")"; }
  private:
    float x, y, z;
};

int main()
{ Point3D a(3,5,7);

  cout << "a = ";
  a.print();
  cout << endl;

  cout << "r = " << a.norm() << endl;
}
```

```
[panos@pc139 ~]$ c++ point3D.cpp
[panos@pc139 ~]$ a.out
a = (3,5,7)
r = 9.11043
[panos@pc139 ~]$
```