OO Programming

N. Cavallini

Procedural vs OOP

Procedural

- A monolithic sequence of data and instructions acting on that data.
- Subroutines and functions are acting on the data.
- Project size ~= 20k lines.

Object Oriented

- Data and functions are collected in Objects.
- Objects interact and communicate with each other.
- Project size ~= 150k lines.

“Write programs that do one thing and do it well. Write programs to work together.”

Unix Philosophy.

Procedural vs OOP

object_oriented.py

```python
import numpy as np
import matplotlib.pyplot as plt

from func_lib import *

x = np.linspace(0,3,100)
y = np.array([])

p1 = Parabola(1,3,2)
p1.evaluate(x,y)
p1.plot()

p2 = Parabola(1,-3,2)
p2.evaluate(x,y)
p2.plot()

p3 = Parabola(4,-16.8,17.6)
p3.evaluate(x,y)
p3.plot()

e1 = Ellipse(1,20,1.5)
e1.evaluate(x,y)
e1.plot()

plt.show()
```

procedural.py

```python
import numpy as np
import matplotlib.pyplot as plt

a = 1
b = 3
c = 2

x = np.linspace(0,3,100)
p1 = a*x**2 + b*x + c

a = 1
b = -3
c = 2

p2 = a*x**2 + b*x + c

a = 4
b = -16.8
c = 17.6

p3 = a*x**2 + b*x + c


plt.plot(x,p1)
plt.plot(x,p2)
plt.plot(x,p3)

a = 1.
b = 20

e1 = 1.-(x/a)**2
domain = np.where(e1>0)
e1 = b*np.sqrt(e1[domain])

plt.plot(x[domain],e1)
plt.grid()
plt.show()
#
# For some reason i need here the p1 coefficents...
```

OO Languages

MHPC
Master in High Performance Computing

julia

Fortran

C

C++

python

OO Support

OO Languages

OO Support

OO Languages

Master in High Performance Computing

julia

Rapidly, (extremely rapidly) growing
high level language
dedicated, to HPC in Scientific Computing.

OO Support

OO Languages

WARNING: a modernly designed language dedicated to scientific computing doesn’t support OOP??!?!

OO Languages

Master in High Performance Computing

julia

$$\text{OOP} \neq \text{modern}$$

OO Support

OO Languages

Master in High Performance Computing

julia

$$\text{OOP} \neq \text{magic wand}$$

OO Support

OO Languages

Master in High Performance Computing

julia

$$\text{OOP} = \text{dangerous}$$

OO Languages

$$\mathrm{OOP} = \mathrm{dangerous}$$

Mathematics is our tool to solve problems.

OO Support

OO Languages

$$\mathrm{OOP} = \mathrm{dangerous}$$

Mathematics is our tool to solve problems.
To solve problems we develop an abstract model of realty.

OO Support

OO Languages

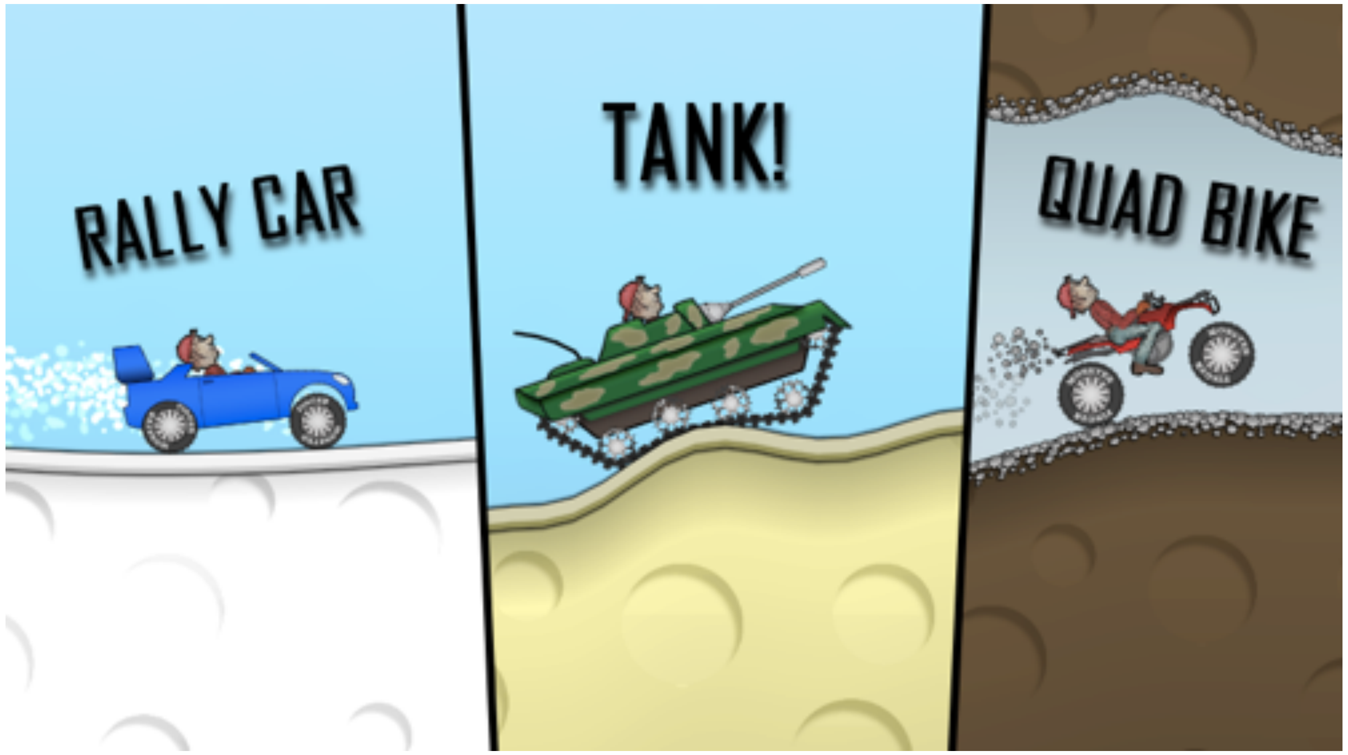
$$\mathrm{OOP} = \mathrm{dangerous}$$

Mathematics is our tool to solve problems.
To solve problems we develop an abstract model of realty.
The more you abstract from reality, the more difficult is to recognise Objects.

Basic Concepts

Easy example where we can recognise most of the Objects

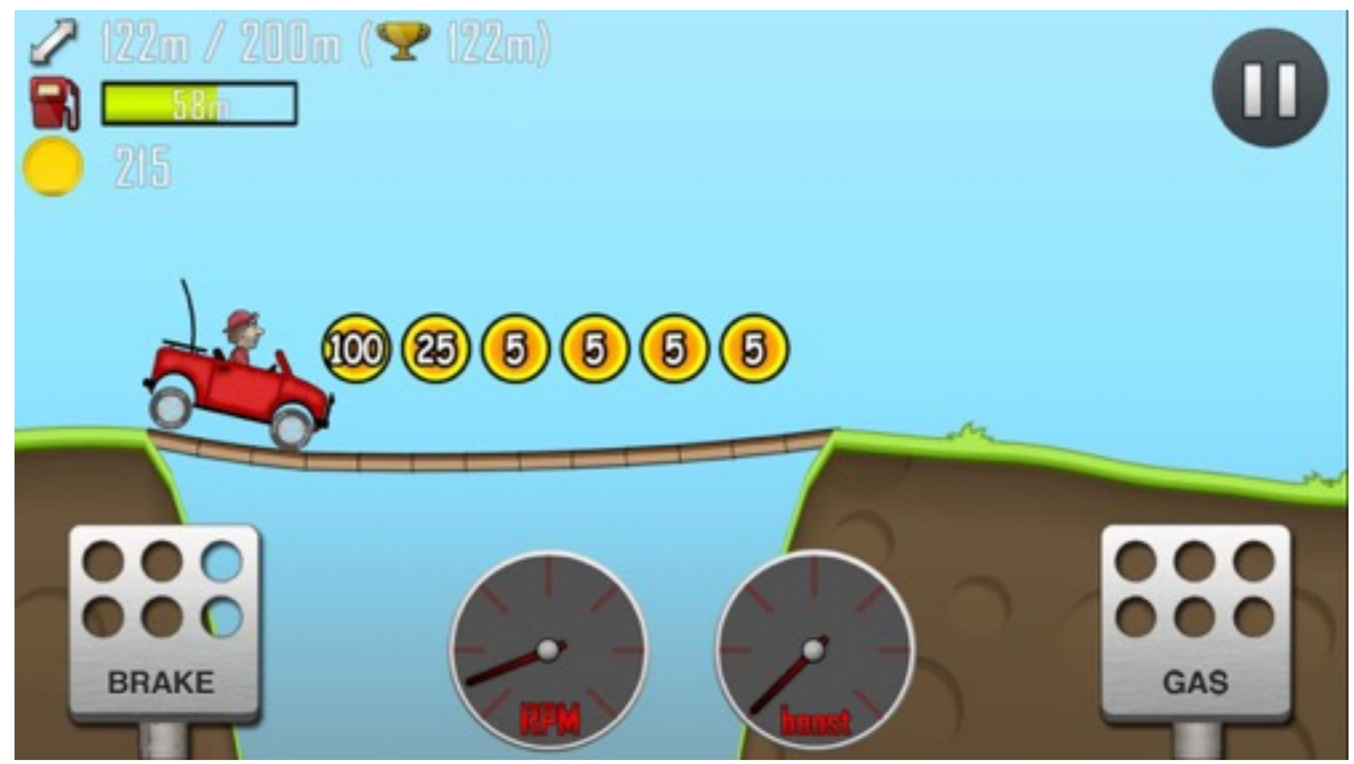
Basic Concepts



Basic Concepts

Object: a collection of data and functions.

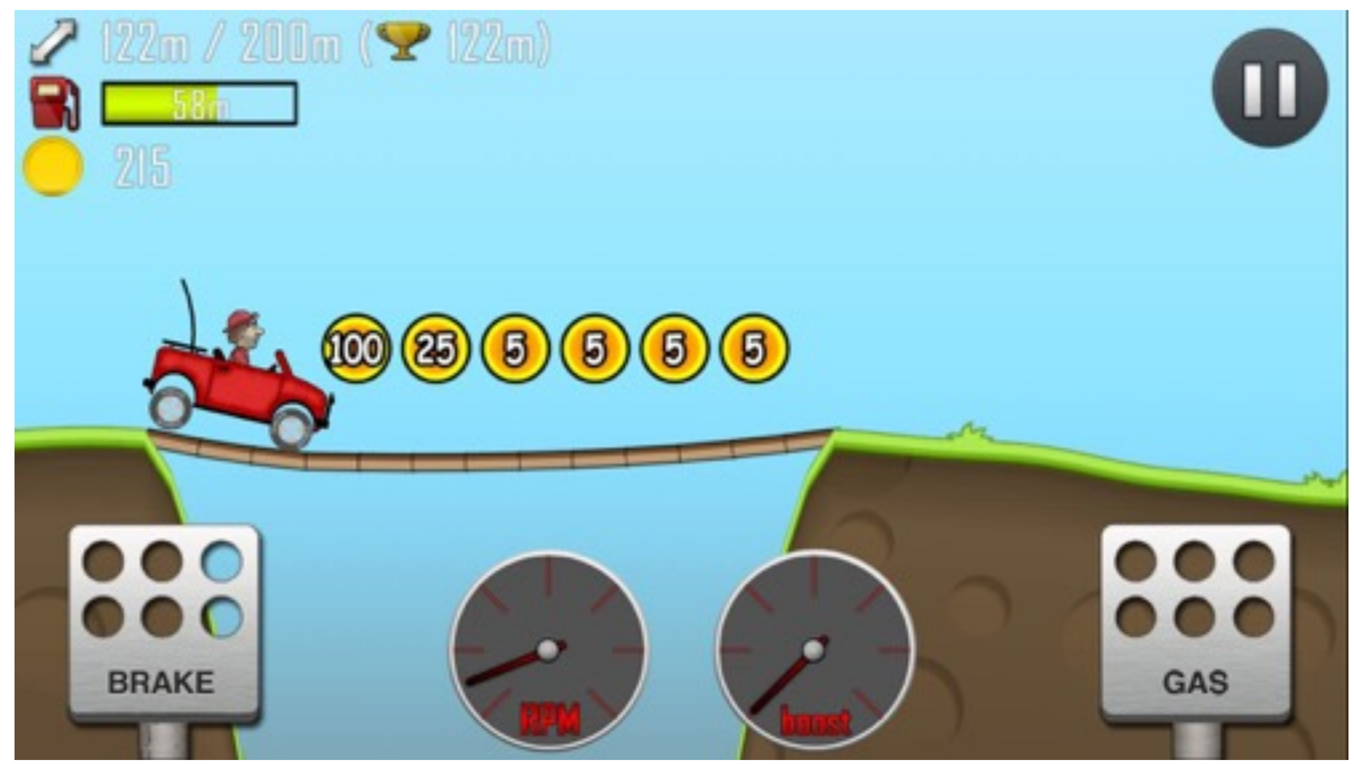
```
object Jeep:

 var EngineCost=4k,
 EnginePower=1kW

 func gas(){}

 func brake(){}
```



Basic Concepts

Object: a collection of data and functions.

```
object Jeep:

 var EngineCost=4k,
 EnginePower=1kW

 func gas(){}

 func brake(){}
```

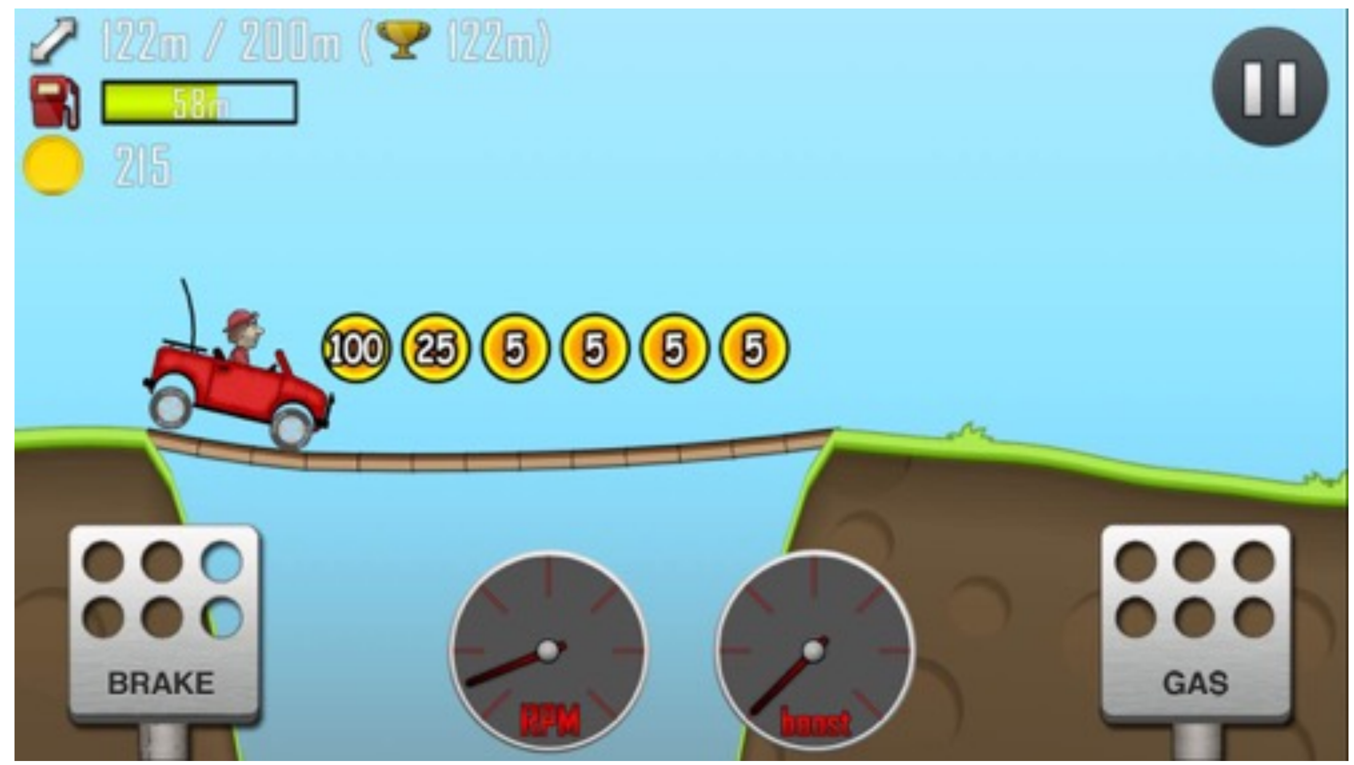


Object *Method*, or
Member Function

Basic Concepts

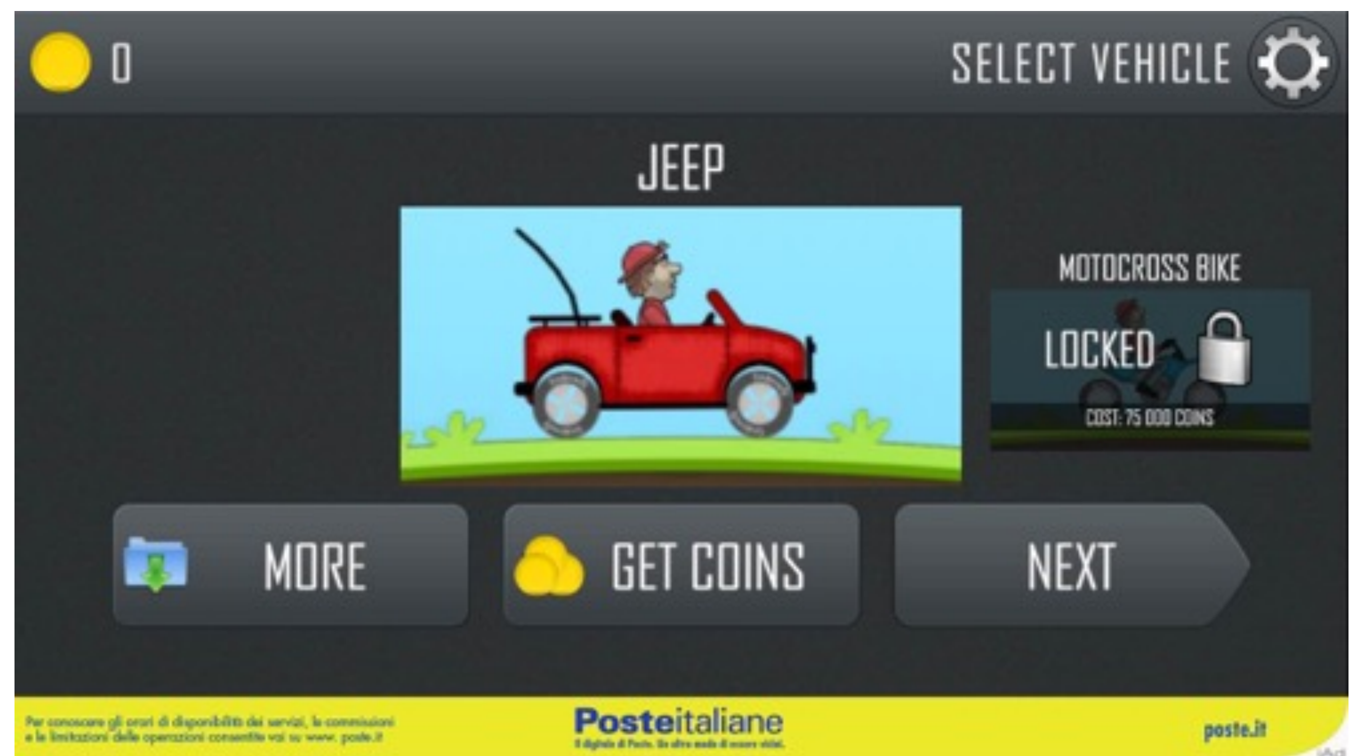
Object: a collection of data and functions.

```
object Jeep:

var EngineCost=4k,
EnginePower=1kW

func gas(){}

func brake(){}
```



Object *Attributes*, or
Member Variables

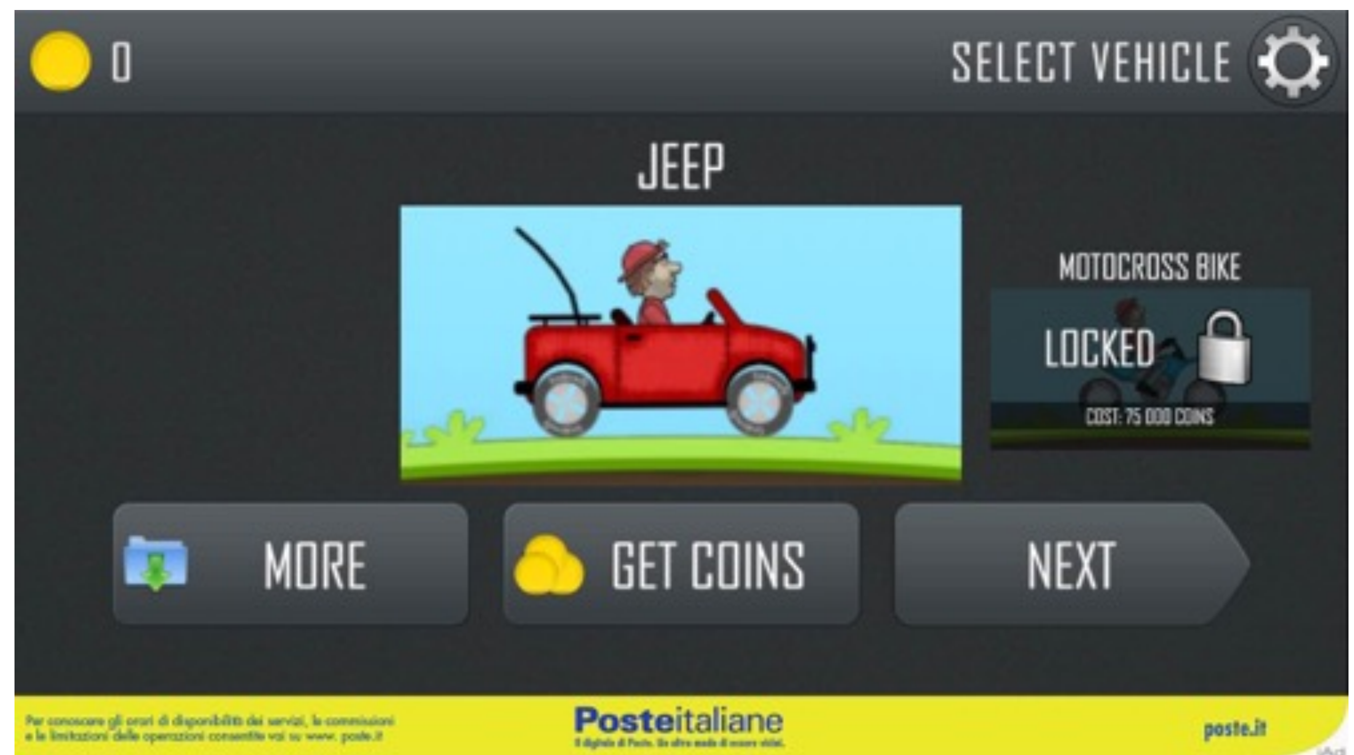
Basic Concepts



MHPC
Master in High Performance Computing

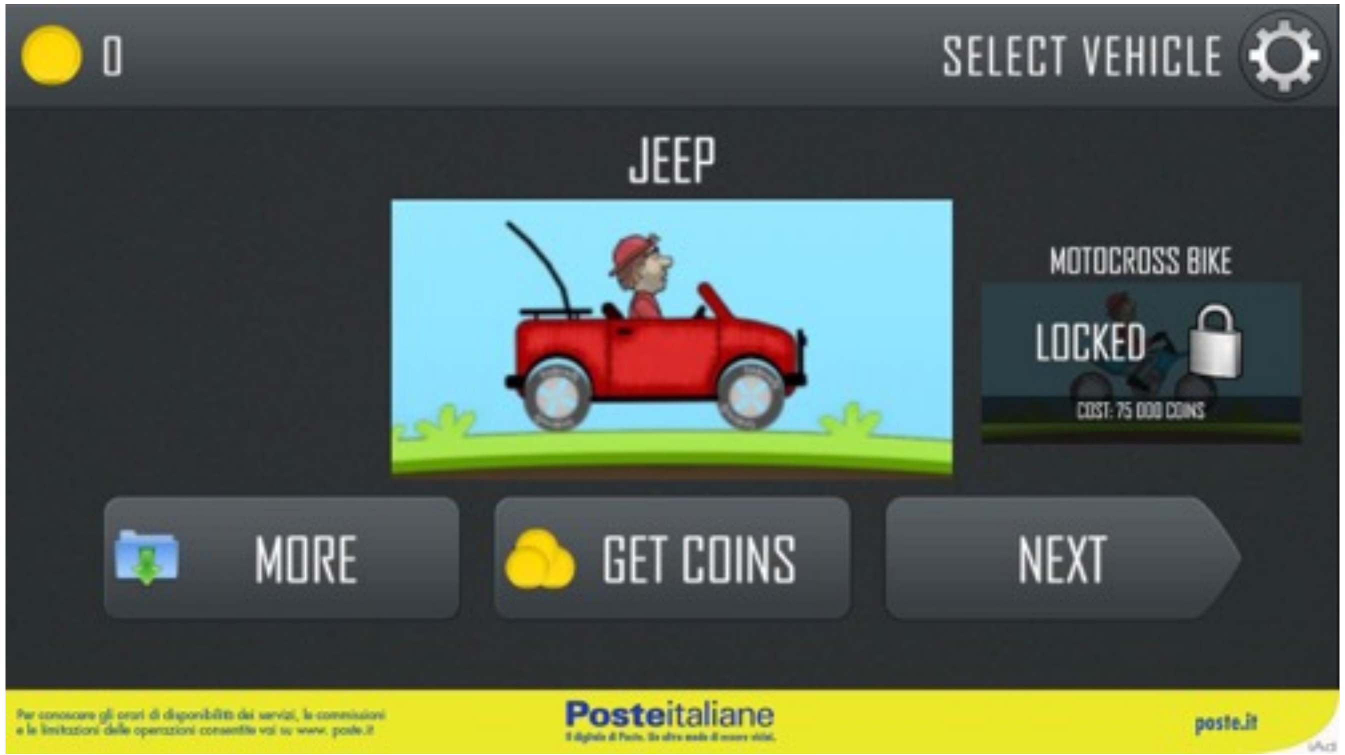
Basic Concepts

```
object Vehicle:

object Jeep:
```



Basic Concepts

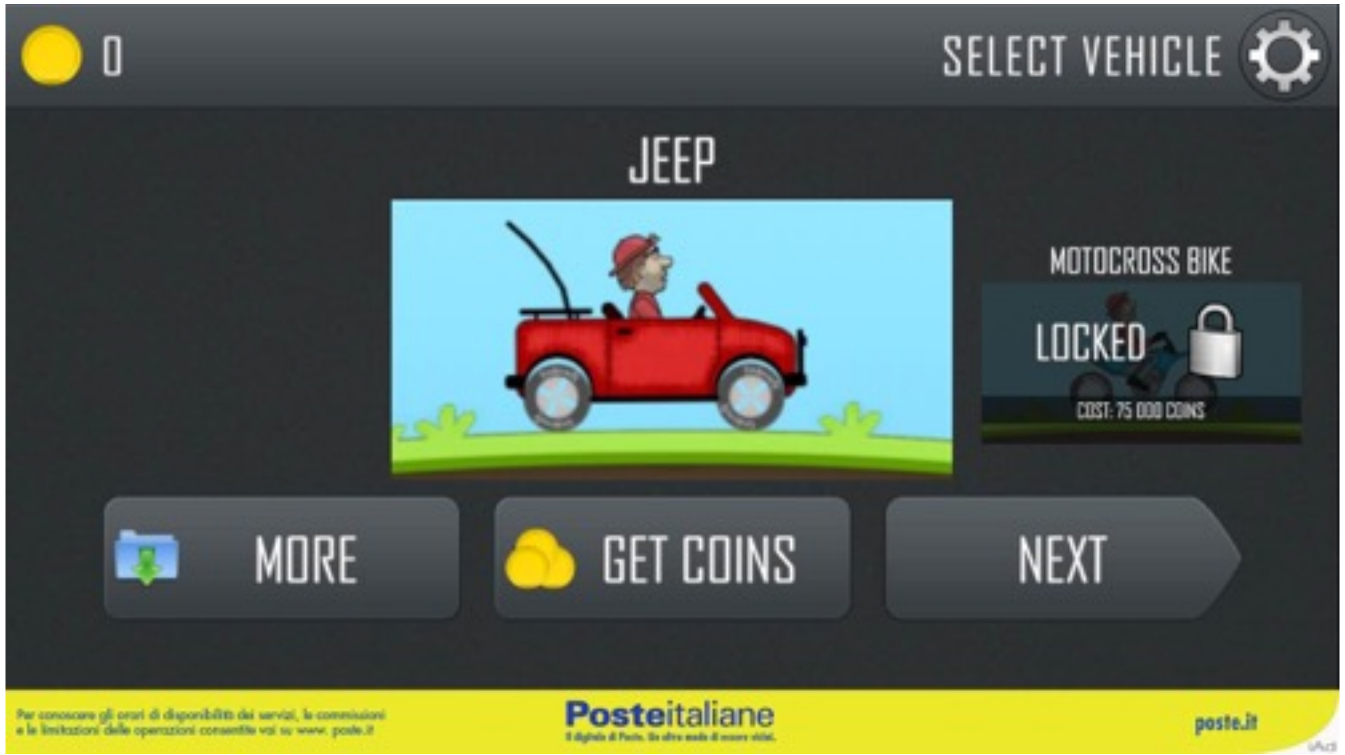
```
object Vehicle:

object Jeep:
```



Parent or *Base* Object

Basic Concepts

```
object Vehicle:

object Jeep:
```



Derived Object

Basic Concepts

Inheritance, we will see three features about it:

- Variables specification.
- Method inherited from Base Object.
- Derived Object specific Method.

Basic Concepts

```
object Vehicle:

 var EngineCost,
 EnginePower

object Jeep:

 var EngineCost=4k,
 EnginePower=1kW
```

Basic Concepts

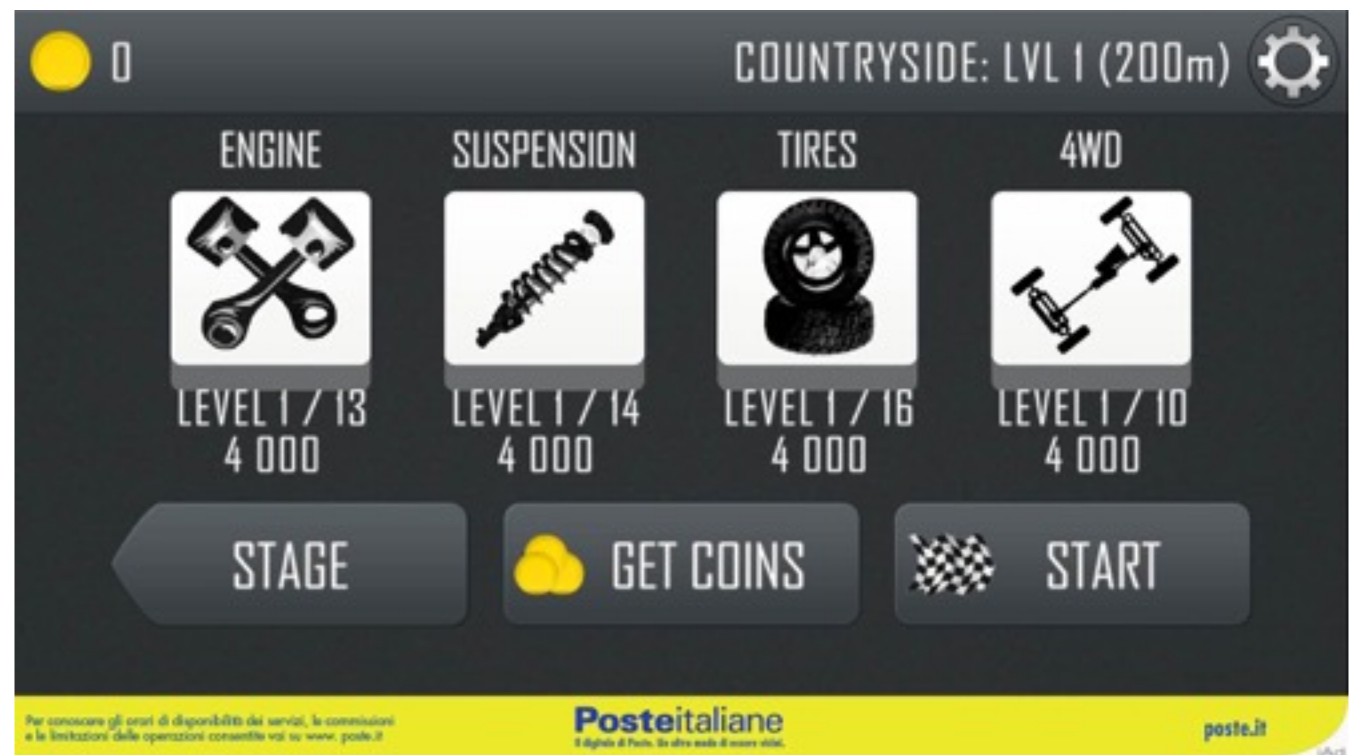
```
object Vehicle:
```

```
var EngineCost,
EnginePower
```

```
object Jeep:
```

```
var EngineCost=4k,
EnginePower=1kW
```

Object *Attributes*, or
Member Variables



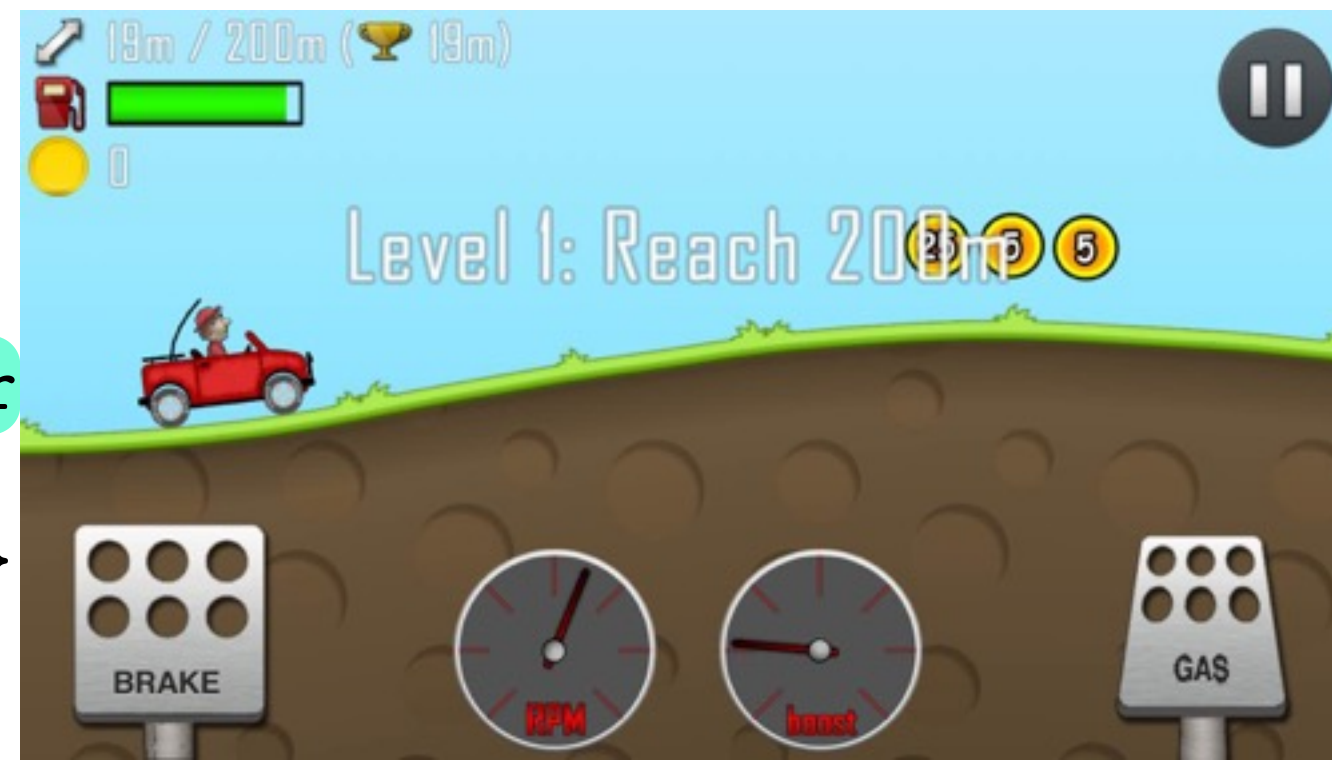
Basic Concepts

```
object Vehicle:

 var EngineCost,
 EnginePower

 func gas(){}

object Jeep:

 var EngineCost=4k,
 EnginePower=1kW
```

Every vehicle is equipped with the `gas` method.

Basic Concepts

```
object Vehicle:

 var EngineCost,
 EnginePower

 func gas(){

 return acc = EnginePower

 *TimeSpentPushingButton}

object Jeep:

 var EngineCost=4k,
 EnginePower=1kW
```

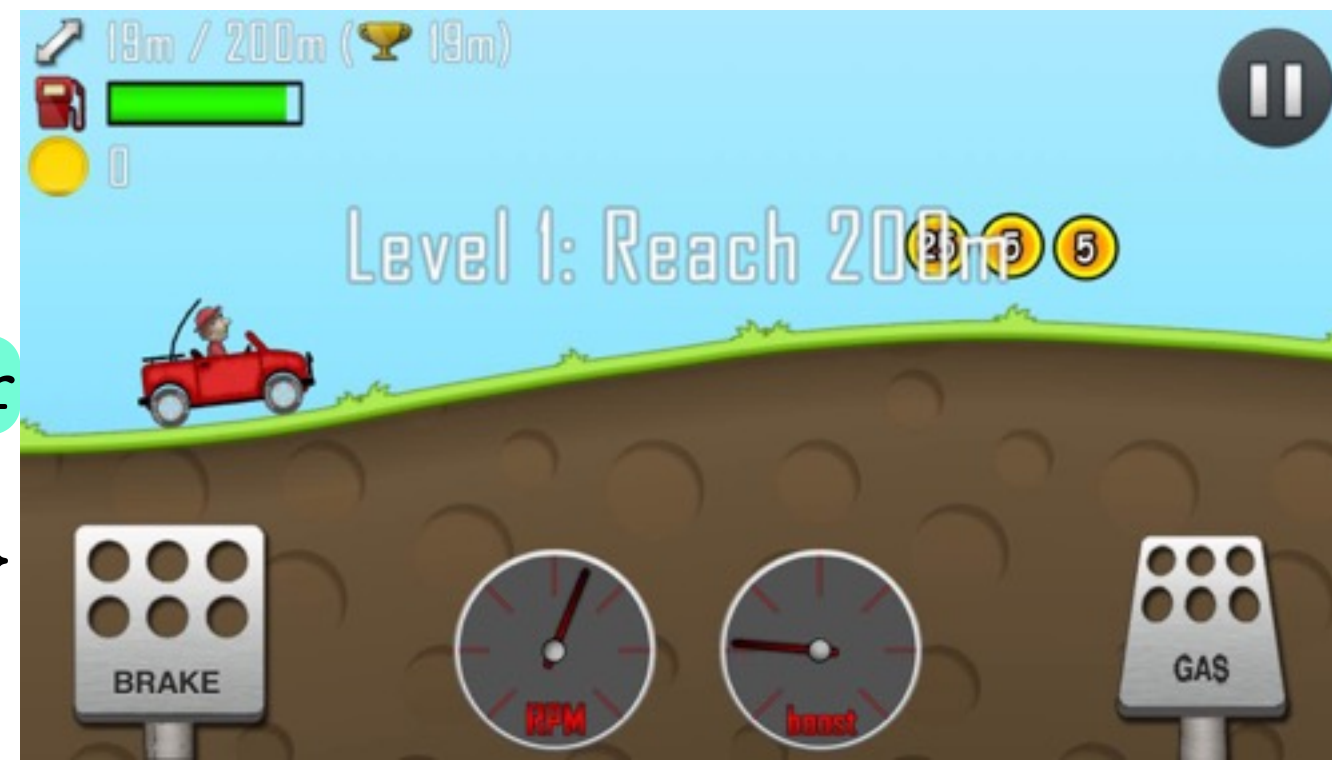


Every vehicle is equipped with the `gas` method.

Master in High Performance Computing

Basic Concepts

```
object Vehicle:

 var EngineCost,
 EnginePower

 func gas(){

 return acc = EnginePower

 *TimeSpentPushingButton}

object Jeep:

 var EngineCost=4k,
 EnginePower=1kW
```



The **gas** method is going to be a **Jeep** method as well

Basic Concepts

```
object Vehicle:

 var EngineCost,
 EnginePower

 func gas(){}

object Jeep:

 func jump(){}

 var EngineCost=4k,
 EnginePower=1kW
```

Say that the **`Jeep`** class can jump, but other **`Vehicles`** can not.

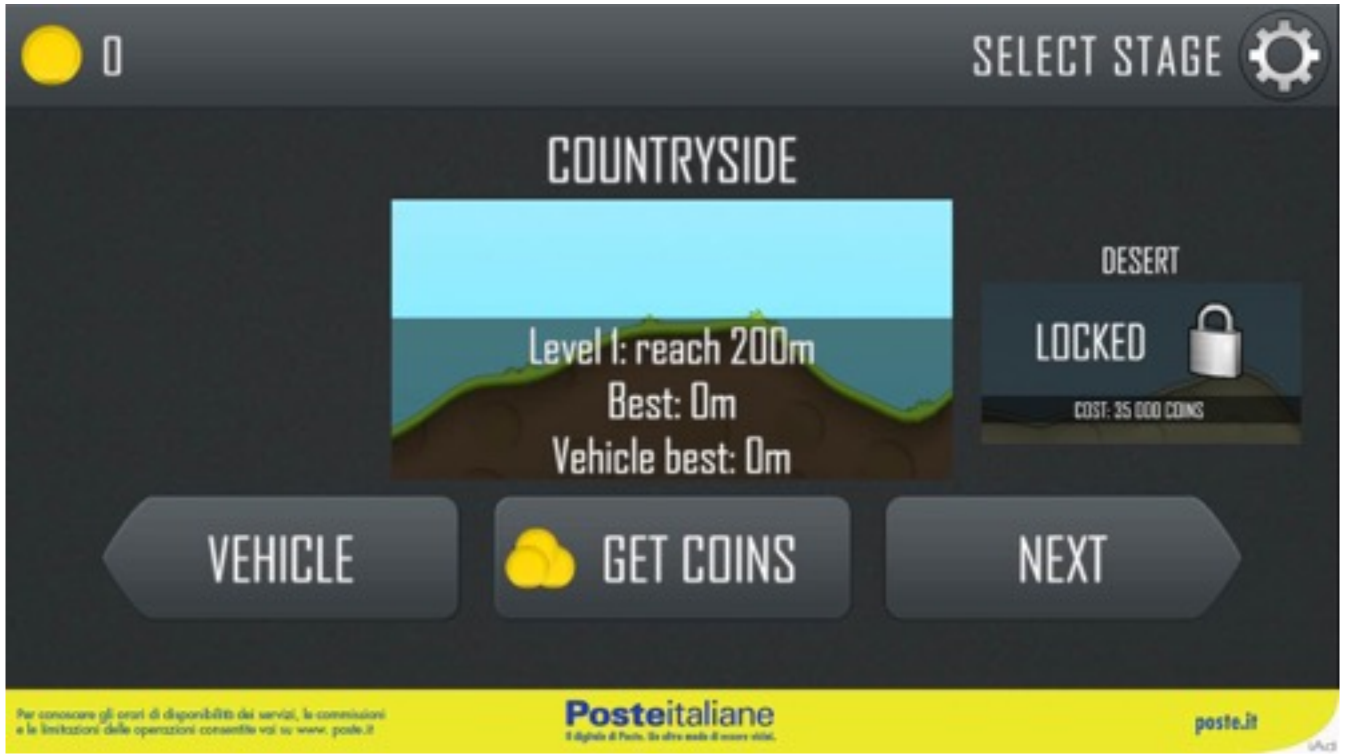
Basic Concepts

```
object Vehicle:

 var EngineCost,
 EnginePower

 func gas(){}

object Jeep:

 func jump(){}

 var EngineCost=4k,
 EnginePower=1kW
```

The **`jump`** method is going to be a **`Jeep`** method but not a **`Vehicle`** method.

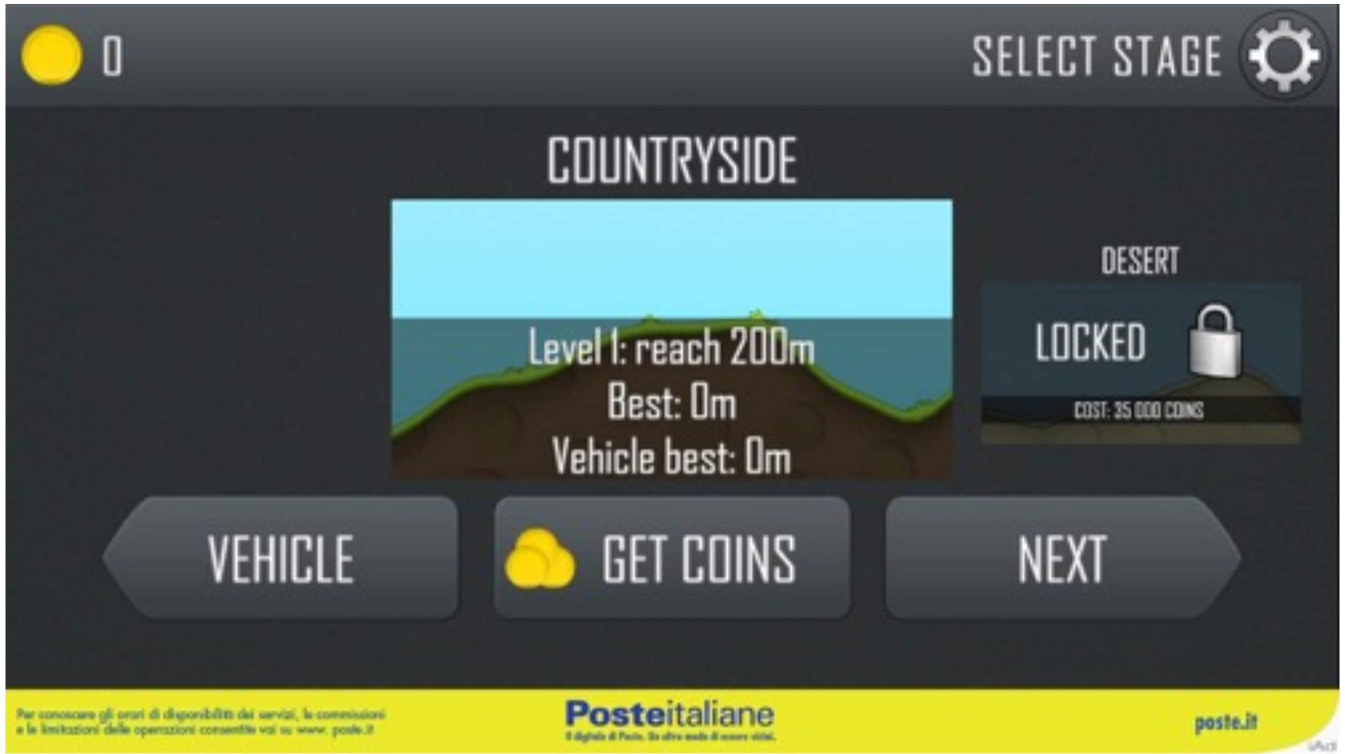
Basic Concepts

```
object Stage:

object Countryside:
```



Basic Concepts

```
object Stage:

 var Level, Best, VBest

object Countryside:

 var Level=200, Best=0,
 VBest=0
```



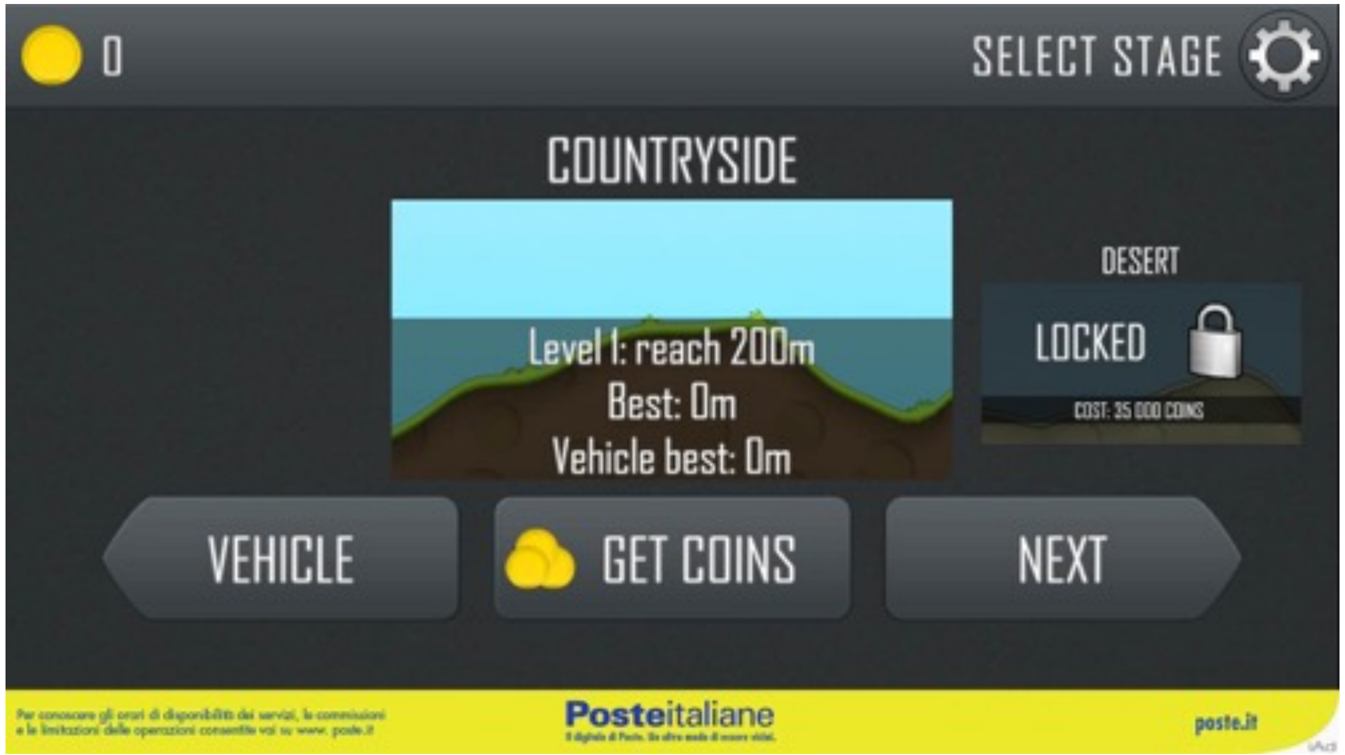
Basic Concepts

```
object Stage:
```

```
var Level, Best, VBest
```

```
object Countryside:
```

```
var Level=200, Best=0,
VBest=0
```

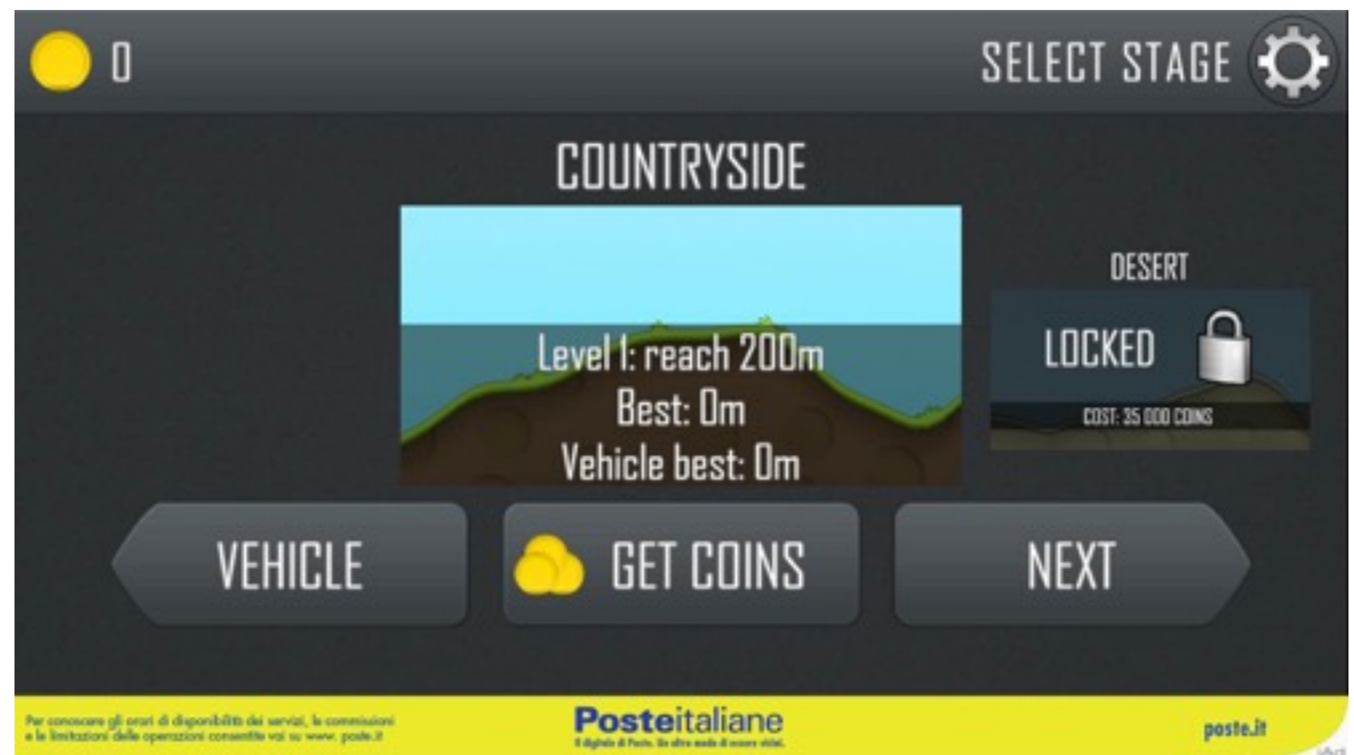


Object *Attributes*, or
Member Variables

MHPC Master in High Performance Computing

Basic Concepts

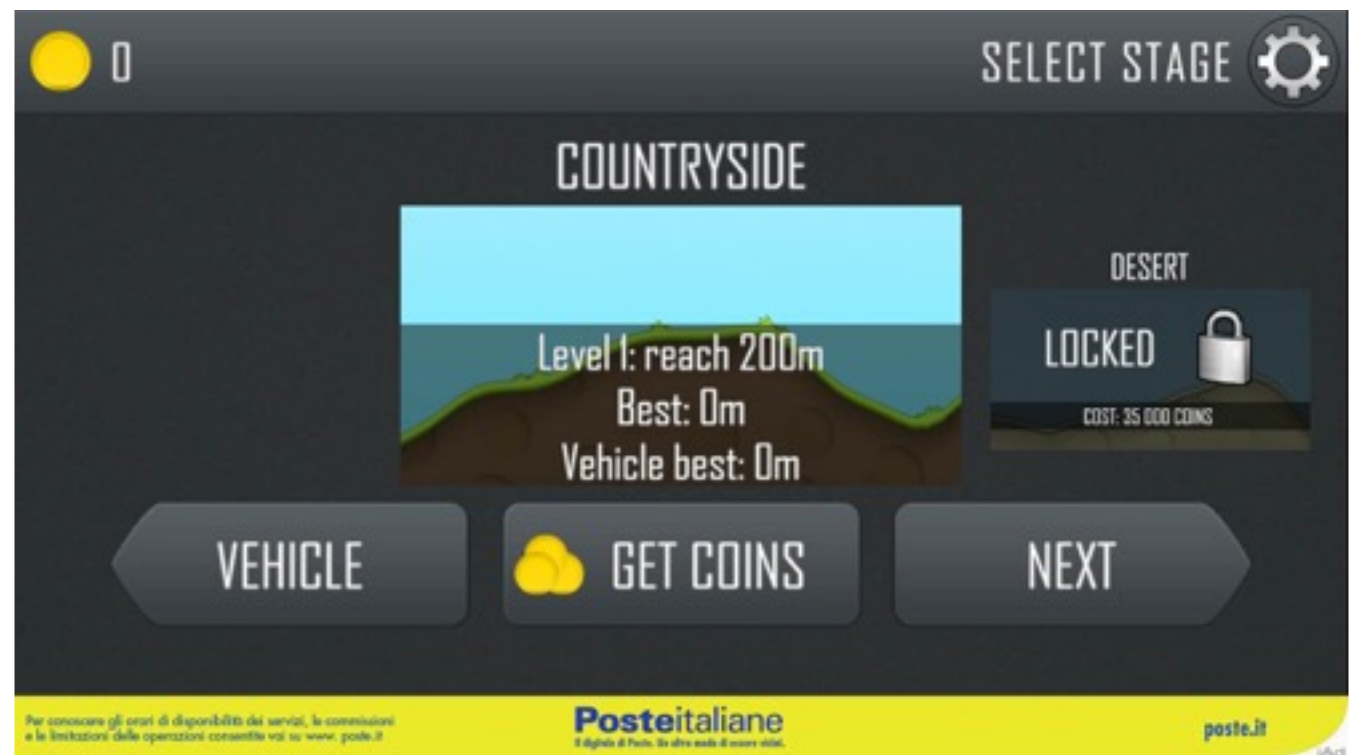
```
object Stage:

 var Level, Best, VBest

 func slope(point){

 "not implemented"}

object Countryside:

 var Level=200, Best=0,
 VBest=0

 func slope(point){

 return point^2}
```



Methods That you cannot specify at the base level are called Virtual

Basic Concepts

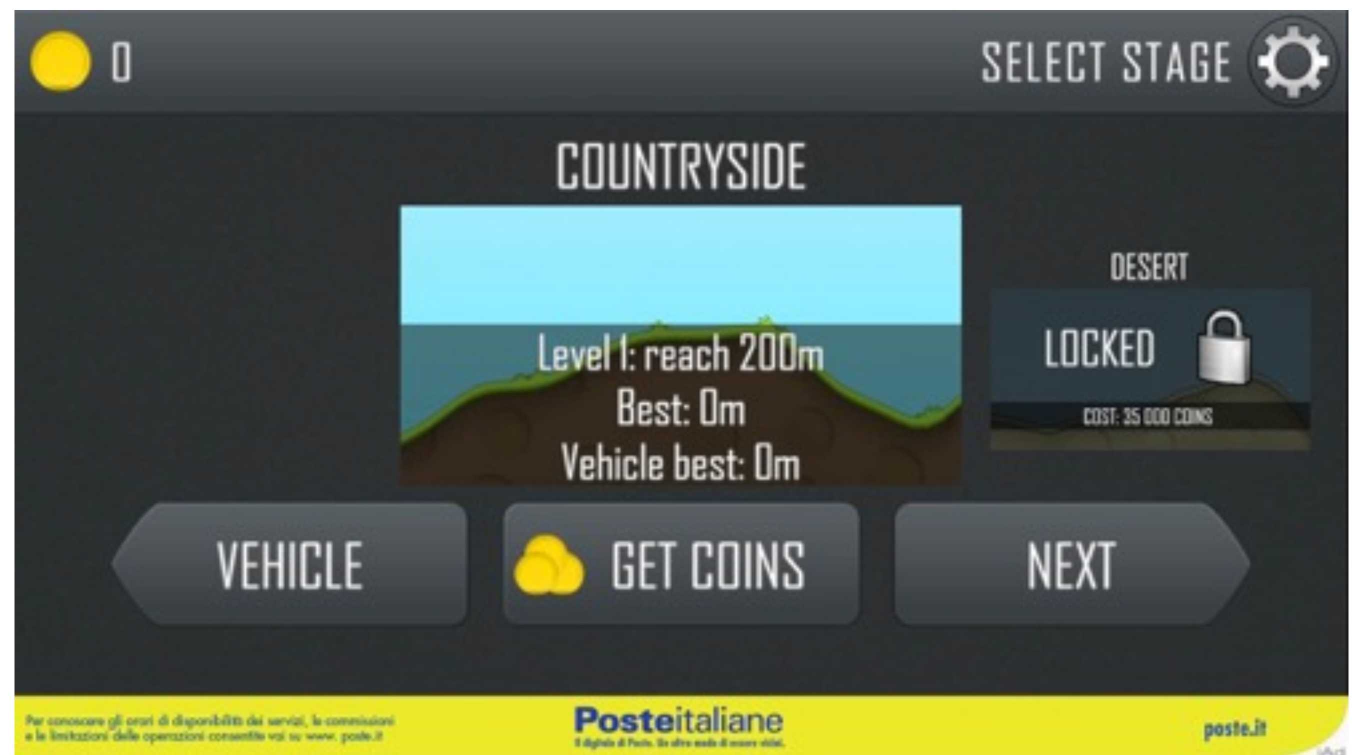
```
object Stage:

 var Level, Best, VBest

 func slope(point){

 "not implemented"}

object Countryside:

 var Level=200, Best=0,
 VBest=0

 func slope(point){

 return point^2}
```



Please forgive me: Think about them as place holders.

Basic Concepts

```
object Stage:

 var Level, Best, VBest

func slope(point){

“not implemented”}

object Countryside:

 var Level=200, Best=0,
 VBest=0

func slope(point){

return point^2}
```



Please forgive me: Think about them as place holders.

Basic Concepts

Now we see the two objects *Acting* together.

```
object Action:

func setup(){

  }
```

Basic Concepts

Now we see the two objects *Acting* together.

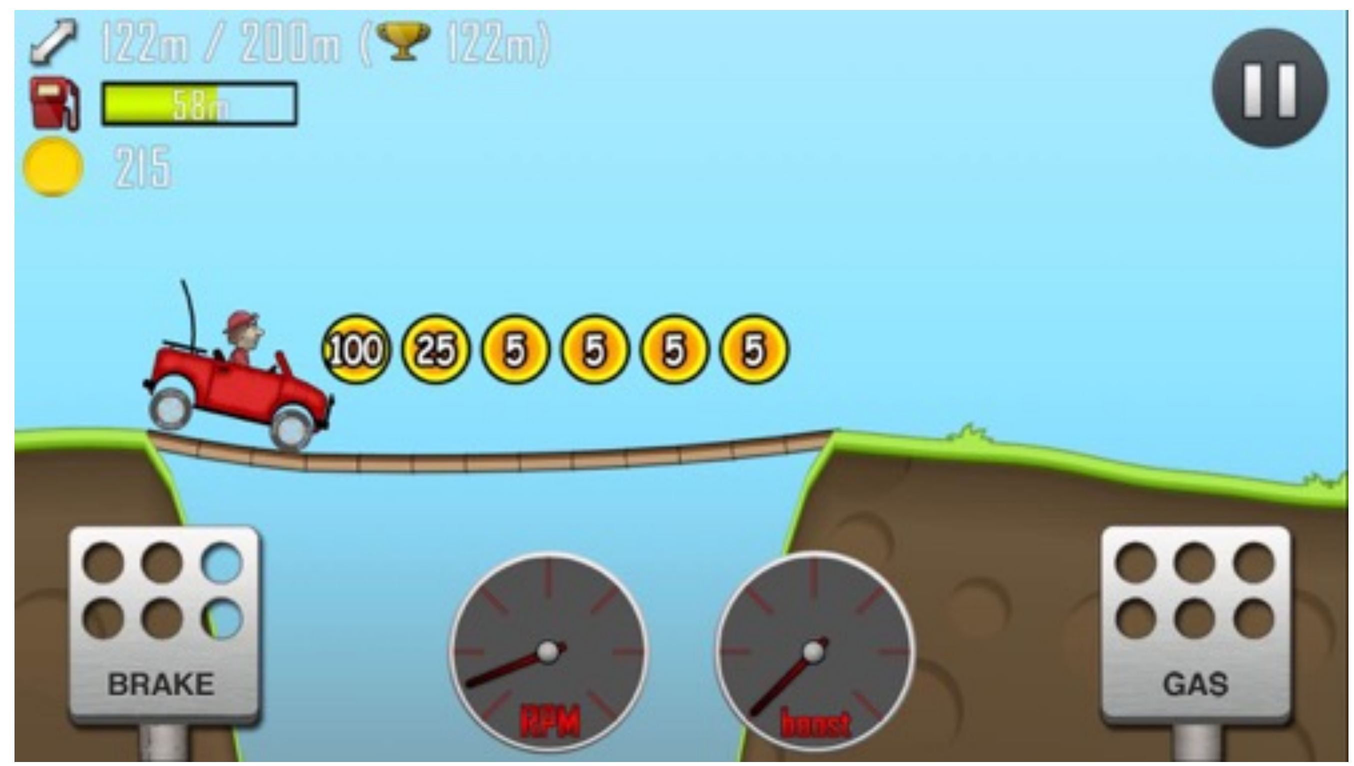
```
object Action:
```

```
func setup(){

  }
```

Object *Method*, or
Member Function

Basic Concepts

```
object Action:

func setup{

 Jeep jeep_0

 Countryside cs_0}

func run(){

  }
```



In the object **`Action`**, we have *instances* **`jeep_0`** and **`cs_0`** of **`Jeep`** and **`Countryside`**, respectively.

Basic Concepts

```
object Action:

func setup{

 Jeep jeep_0

 Countryside cs_0}

func run(){

  }
```

In the object `Action`, is a *composition* of `Jeep` and `Countryside`.

```
object Action:

func setup(){

 Jeep jeep_0

 Countryside cs_0}

func run(){

 acc = jeep_0.gas()

 jeep_0.vel(acc,
 cs_0.slope())

 }
```

A common syntax to call methods and attributes, is the dot.

```
object Action:

func setup(){

 Jeep jeep_0

 Countryside cs_0}

func run(){

 acc = jeep_0.gas()

 jeep_0.vel(acc,
 cs_0.slope())

 }
```

The `jeep_0` velocity is the result of the acceleration, and `cs_0` slope.

```
object Action:

func setup(){

 Jeep jeep_0

 Countryside cs_0}

func run(){

 acc = jeep_0.gas()

 jeep_0.vel(acc,
 cs_0.slope())

 }
```

The **`jeep_0`** velocity is the result of the acceleration, and **`cs_0`** slope.

```
object Action:

func setup(){

  Jeep jeep_0

  Countryside cs_0}

func run(){

  acc = jeep_0.gas()

  jeep_0.vel(acc,
  cs_0.slope())}

main(){

  Action act_0;

  act_0.setup()

  act_0.run()

  }
```

Instantiation

MHPC
Master in High Performance Computing

```
object Action:

func setup(){

  Jeep jeep_0

  Countryside cs_0}

func run(){

  acc = jeep_0.gas()

  jeep_0.vel(acc,
  cs_0.slope())}

main(){

  Action act_0;

  act_0.setup()

  act_0.run()

  }
```

Instantiation

Basic Concepts

Introduced concepts:

- Object
- Method - Member Function
- Attribute - Member Variable
- Base Object
- Derived Object
- Inheritance
- Virtual Functions
- Composition

Basic Concepts

Wrap up:

- An Object is a collection of *member variables* and *member functions*, ***conceptually related with each other***.
- Object can be hierarchically related by *inheritance.*
- We can *compose* objects, making an object a member variable of a second one.

Python Objects

$$f : X \to Y$$

Python Objects

$$f : X \to Y$$

$$\begin{array}{ll} X & \text{domain} \\ Y & \text{codomain} \end{array}$$

Python Objects

$$f : X \to Y$$

X domain

Y codomain

```
class Function:
```

We define our class Function

Python Objects

$$f : X \to Y$$

$$X \quad \text{domain}$$
$$Y \quad \text{codomain}$$

```
class Function:

 def some_method(self,var1)

 return
```

We define our class Function

Python Objects

$$f : X \to Y$$

$$\begin{array}{ll} X & \text{domain} \\ Y & \text{codomain} \end{array}$$

method of argument var1

```
class Function:

 def some_method(self,var1):

 return
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{ll} X & \text{domain} \\ Y & \text{codomain} \end{array}$$

`self`
Has exactly the same meaning as `this` in `c++`

```
class Function:

 def some_method(self,var1):

 return
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{ll} X & \text{domain} \\ Y & \text{codomain} \end{array}$$

It is a reference to the instantiated object.

```
class Function:

 def some_method(self,var1):

 return
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{ll} X & \text{domain} \\ Y & \text{codomain} \end{array}$$

```
class Function:

 def some_method(self,var1):

 return

func1 = Function()

func1.some_method(1.0)
```

It is a reference to the instantiated object.

Object instantiation.

Python Objects

$$f : X \to Y$$

$$\begin{array}{ll} X & \text{domain} \\ Y & \text{codomain} \end{array}$$

```
class Function:

 def some_method(self,var1):

 return

func1 = Function()

func1.some_method(1.0)
```

It is a reference to the instantiated object.

Method call

Python Objects

$$f : X \to Y$$

$$\begin{array}{ll} X & \text{domain} \\ Y & \text{codomain} \end{array}$$

```
class Function:

 def some_method(self,var1):

 return

func1 = Function()

func1.some_method(var1)
```

It is a reference to the instantiated object.

you shall read this as:

```
func1.some_method(func1,var1)
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

We construct a function and assign it a pointer. Named **`function`**.

```
function = Function()
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

`function`, this is also a python type… Don't try this at home :)

```
function = Function()
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

Python syntax for a *constructor.*

```
class Function:

 def __init__(self):

  return
```

The method to construct a Class in python is called *constructor.*

```
function = Function()
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

Python syntax for a *constructor.*

```
class Function:

 def __init__(self):

  return
```

The method to construct a Class in python is called *constructor.*

```
function = Function()
```

Python Objects

Now we equipe our class with variables that might be useful in several class methods.

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()
```

Python Objects

Master in High Performance Computing

$$f : X \to Y$$

What variables?
Design!

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:
  def __init__(self):
    self.dim_domain = 1
    self.dim_codomain = 1
    self.domain = np.array([])
    self.codomain = np.array([])
    return
```

```
function = Function()
```

Python Objects

There should exist a one to one correspondence in between mathematics and object design

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()
```

Python Objects

There should exist a one to one correspondence in between mathematics and object design

$$f : X \to Y$$

X domain $\subset \mathbb{R}^m$

Y codomain $\subset \mathbb{R}^n$

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()
```

Python Objects

There should exist a one to one correspondence in between mathematics and object design

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()
```

Python Objects

$$f : X \to Y$$

These are public variables to our class.

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()
```

Python Objects

$$f : X \to Y$$

These are public variables to our class.

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()

print(function.dim_domain)

print(domain)
```

Public variables can be accessed outside of the class.

Python Objects

$$f : X \to Y$$

These are public variables to our class.

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()

print(function.dim_domain)

print(domain)
```

In python all methods and attributes are public.

Python Objects

$$f : X \to Y$$

These are public variables to our class.

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()

print(function.dim_domain)

print(domain)
```

C++ allows to distinguish between public and private.

Master in High Performance Computing

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

These are public variables to our class.

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()

print(function.dim_domain)

print(domain)
```

public: accessible outside the class.
private: accessible inside the class only.

Python Objects

$$f : X \to Y$$

These are public variables to our class.

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

  def __init__(self):

    self.dim_domain = 1

    self.dim_codomain = 1

    self.domain = np.array([])

    self.codomain = np.array([])

    return
```

```
function = Function()

print(function.dim_domain)

print(function.domain)
```

Convention: in python private members are called `__member` to mark them as private.

Python Objects

$$f : X \to Y$$

These are private variables to our class.

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

 def __init__(self):

  self.__dim_domain = 1

  self.__dim_codomain = 1

  self.__domain = np.array([])

  self.__codomain = np.array([])

  return
```

```
function = Function()

print(function.dim_domain)

print(function.domain)
```

Convention: in python private members are called **`__member`** to mark them as private.

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

 def __init__(self):

   self.dim_domain = 1

   self.dim_codomain = 1

   self.domain = np.array([])

   self.codomain = np.array([])

   return
```

```
function = Function()

print(function.dim_domain)

print(function.domain)
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

This is going to be our base class.

```
class Function:

 def evaluate(self):

  return
```

```
function = Function()

function.evaluate()
```

Python Objects

$$f : X \to Y$$

We will specialise this method as soon as we develop the derived class

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

 def evaluate(self):

  raise NotImplementedError()

  return
```

```
function = Function()

function.evaluate()
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

This will be called *Virtual* function.

```
class Function:

 def evaluate(self):

  raise NotImplementedError()

  return
```

```
function = Function()

function.evaluate()
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

This will be called *Virtual* function.

```
class Function:

 def evaluate(self):

  raise NotImplementedError()

  return
```

```
function = Function()

function.evaluate()
```

Python Objects

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

We can use them to define the *Interface*

```
class Function:

 def evaluate(self):

  raise NotImplementedError()

  return
```

```
function = Function()

function.evaluate()
```

Python Objects

Interface: Collection of Attributes and methods, that define how Object interact with each other.

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

 def evaluate(self):

  raise NotImplementedError()

  return
```

```
function = Function()

function.evaluate()
```

Python Objects

A method that could be common to all functions could be, plot

$$f : X \to Y$$

$$\begin{array}{lll} X & \text{domain} & \subset \mathbb{R}^m \\ Y & \text{codomain} & \subset \mathbb{R}^n \end{array}$$

```
class Function:

 def plot(self):

 if (self.dim_domain == 1 and
 self.dim_codomain==1):

  plt.plot(self.domain,self.codomain)

 return
```

```
function = Function()

function.plot()
```

Python Objects

$$y = a\,x^2 + b\,x + c$$

$$x \in X \quad \text{domain} \quad \subset \mathbb{R}^1$$
$$y \in Y \quad \text{codomain} \quad \subset \mathbb{R}^1$$

```
class Parabola(Function):

    def __init__(self,a,b,c):

        Function.__init__(self)

        self.a = a

        self.b = b

        self.c = c

        return
```

`Parabola` is a class *derived* from `Function`

Python Objects

$$y = a\,x^2 + b\,x + c$$

$$\begin{array}{lll} x \in X & \text{domain} & \subset \mathbb{R}^1 \\ y \in Y & \text{codomain} & \subset \mathbb{R}^1 \end{array}$$

```
class Parabola(Function):

    def __init__(self,a,b,c):

        Function.__init__(self)

        self.a = a

        self.b = b

        self.c = c

        return
```

As a design decision, we include the coefficients in the constructors.

Python Objects

$$y = a\,x^2 + b\,x + c$$

$$x \in X \quad \text{domain}$$
$$y \in Y \quad \text{codomain}$$

A **`setter`** in this case in an alternative option.

```
class Parabola(Function):

    def __init__(self,a,b,c):

        Function.__init__(self)

        self.a = a

        self.b = b

        self.c = c

        return
```

```
class Parabola(Function):

    def __init__(self):

        return

    def set_cofes(self,a,b,c):

        self.a = a

        self.b = b

        self.c = c

        return
```

Python Objects

Programming wise there in no difference.

```
class Parabola(Function):

    def __init__(self,a,b,c):

        Function.__init__(self)

        self.a = a

        self.b = b

        self.c = c

        return
```

```
class Parabola(Function):

    def __init__(self):

        return

    def set_cofes(self,a,b,c):

        self.a = a

        self.b = b

        self.c = c

        return
```

Python Objects

Conceptually, there is no way to think of a parabola without its coefficients.

```
class Parabola(Function):

    def __init__(self,a,b,c):

        Function.__init__(self)

        self.a = a

        self.b = b

        self.c = c

        return
```

```
class Parabola(Function):

    def __init__(self):

        return

    def set_cofes(self,a,b,c):

        self.a = a

        self.b = b

        self.c = c

        return
```

Python Objects

The constructor is the method that defines the object

```
class Parabola(Function):

    def __init__(self,a,b,c):

        Function.__init__(self)

        self.a = a

        self.b = b

        self.c = c

        return
```

```
class Parabola(Function):

    def __init__(self):

      return

    def set_cofes(self,a,b,c):

        self.a = a

        self.b = b

        self.c = c

        return
```

Python Objects

$$y = a\,x^2 + b\,x + c$$

$$x \in X \quad \text{domain}$$
$$y \in Y \quad \text{codomain}$$

When you have private variables, **`getters`**, make sense.

```
class Parabola(Function):

    def __init__(self,a,b,c):

        Function.__init__(self)

        self.a = a

        self.b = b

        self.c = c

        return
```

```
class Parabola(Function):

    def __init__(self):

      return

    def set_cofes(self,a,b,c):

        self.a = a

        self.b = b

        self.c = c

        return
```

In this way we inherit Base class private variables.

on Objects

$$y = a\,x^2 + b\,x + c$$

$$\begin{array}{lll} x \in X & \text{domain} & \subset \mathbb{R}^1 \\ y \in Y & \text{codomain} & \subset \mathbb{R}^1 \end{array}$$

```
class Parabola(Function):

    def __init__(self,a,b,c):

        Function.__init__(self)

        self.a = a

        self.b = b

        self.c = c

        return
```

In this way we inherit Base class private variables.

...on Objects

$$y = a\,x^2 + b\,x + c$$

$$\begin{array}{llll} x \in X & \text{domain} & \subset \mathbb{R}^1 \\ y \in Y & \text{codomain} & \subset \mathbb{R}^1 \end{array}$$

```
class Parabola(Function):

    def __init__(self,a,b,c):

        Function.__init__(self)

        self.a = a

        self.b = b

        self.c = c

        return
```

```
class Function:

    def __init__(self):

        self.dim_domain = 1

        self.dim_codomain = 1

        self.domain = np.array([])

        self.codomain = np.array([])

        return
```

Python Objects

$$y = a\,x^2 + b\,x + c$$

$$\begin{array}{lll} x \in X & \text{domain} & \subset \mathbb{R}^1 \\ y \in Y & \text{codomain} & \subset \mathbb{R}^1 \end{array}$$

```
class Parabola(Function):

        def evaluate(self,domain,codomain):

        self.domain = …

        self.codomain = …
```

Python Objects

$$y = b\sqrt{1 - \left(\frac{x - s}{a}\right)^2}$$

$$\begin{array}{lll} x \in X & \text{domain} & \subset \mathbb{R}^1 \\ y \in Y & \text{codomain} & \subset \mathbb{R}^1 \end{array}$$

```
class Ellipse(Function):

        def evaluate(self,domain,codomain):

        self.domain = … where …

        self.codomain = …
```

Start with modules

Main file: **main.py**

```
import mymodule

s = mymodule.function_1()

s_e = mymodule.func_2()
```

Module file: **mymodule.py**

```
def function_1():

  return something

def func_2():

  return s_else
```

Start with modules

Main file: **main.py**

```
import mymodule as mm

s = mm.function_1()

s_e = mm.func_2()
```

Module file: **mymodule.py**

```
def function_1():

  return something

def func_2():

  return s_else
```

Start with modules

Main file: **main.py**

```
from mymodule import *

s = function_1()

s_e = func_2()
```

Module file: **mymodule.py**

```
def function_1():

  return something

def func_2():

  return s_else
```

Import Classes

Main file: **`main.py`**

```
from class_file import *

mc = MyClass()
```

Module file: **`class_file.py`**

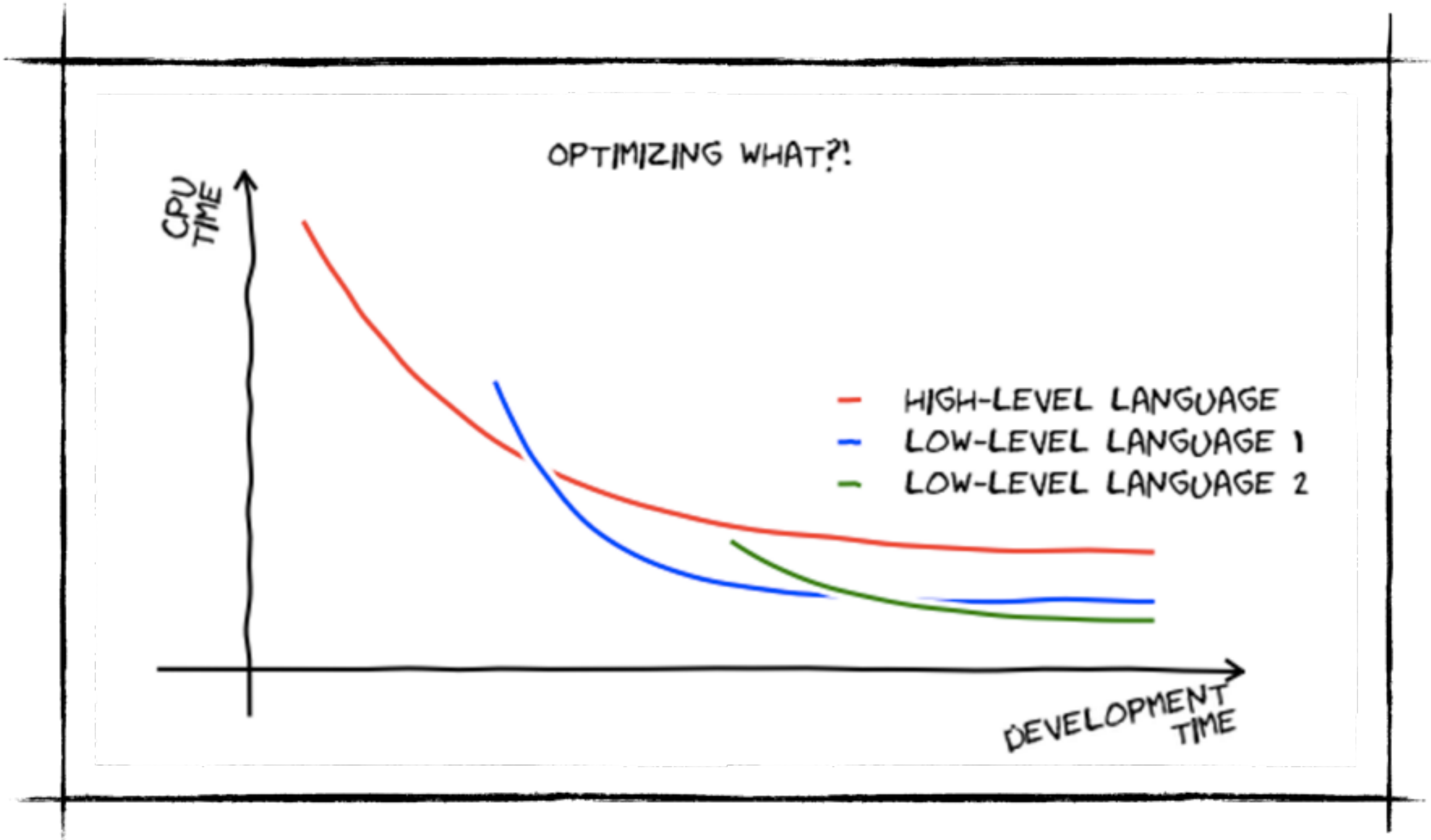
```
class MyClass:

 def function_1(self):

  return something

 def func_2(self):

  return s_else
```

Conclusions

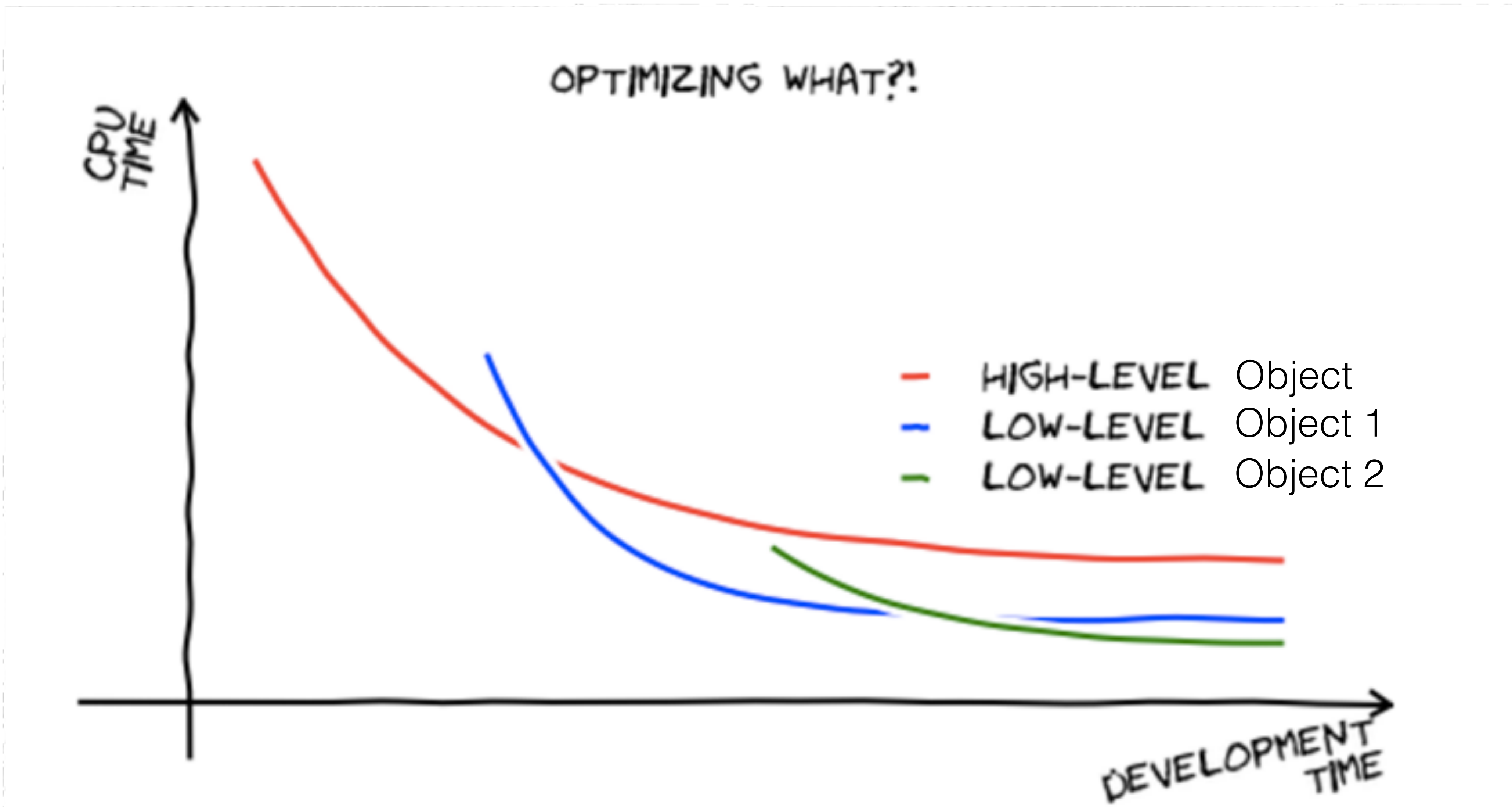
Primary Goal: Scientific Computing.

- Software Development for High Performance ***Correct*** Computing
- Trade off Optimisation, Code readability.

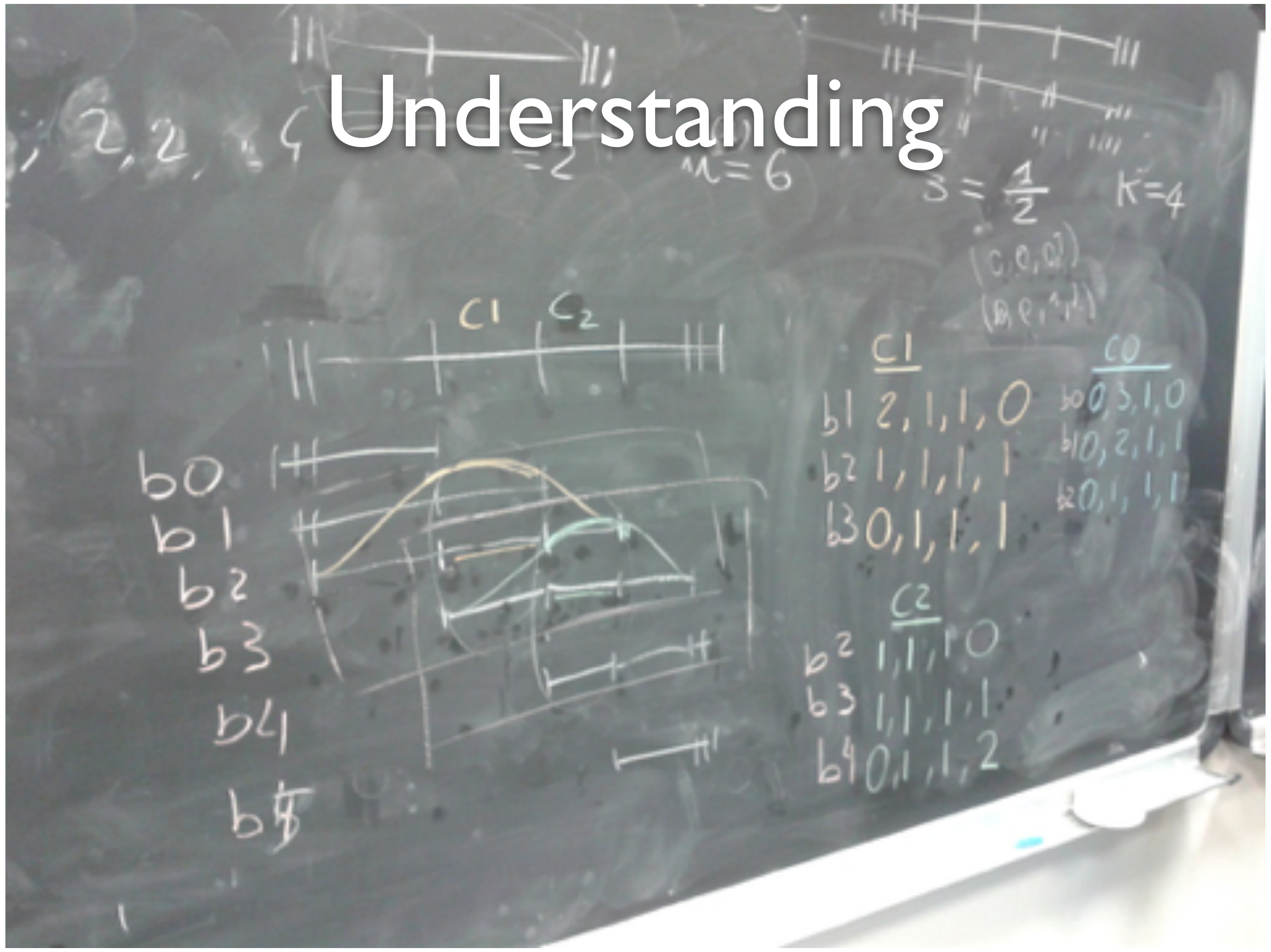
Conclusions

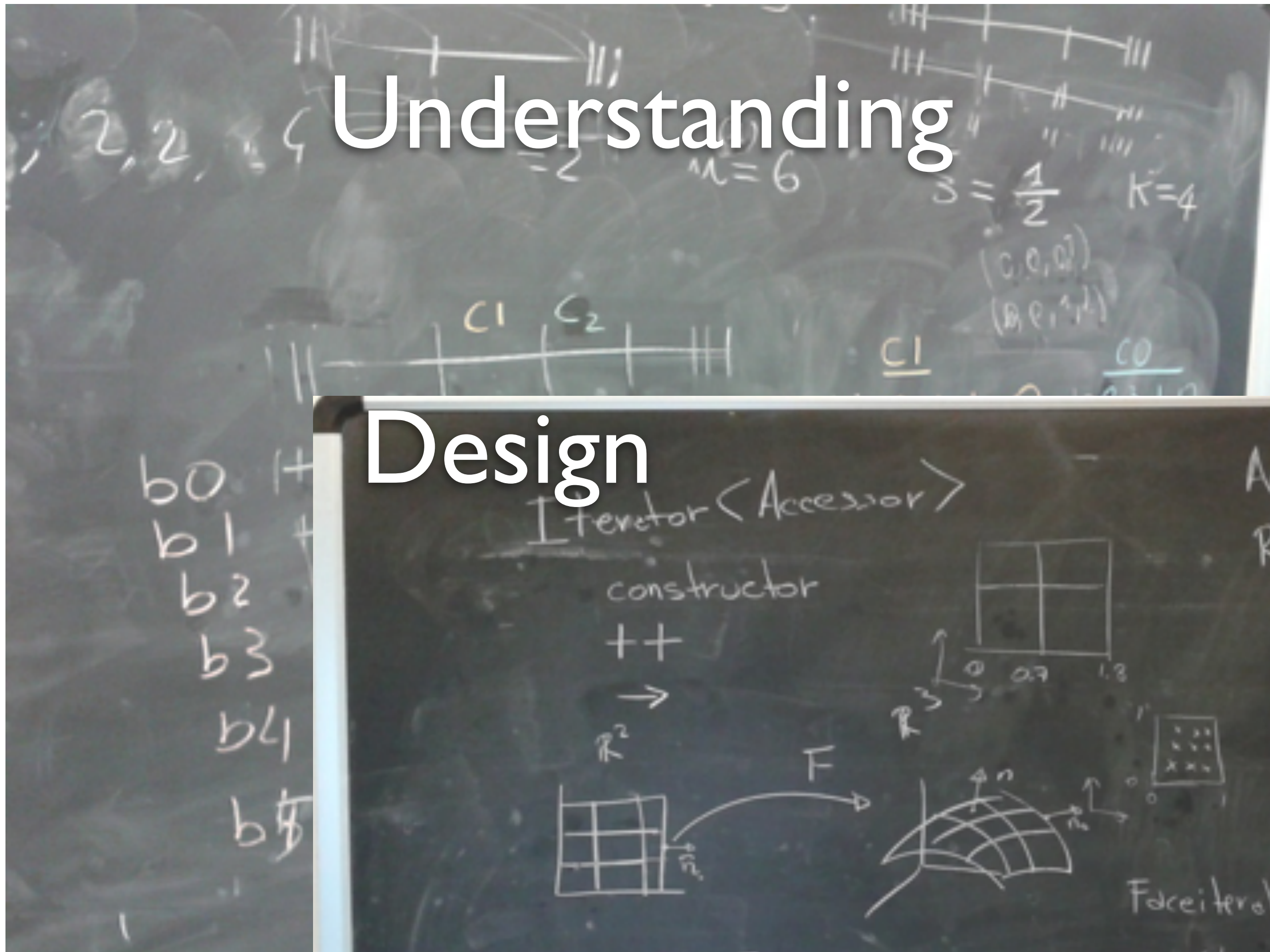


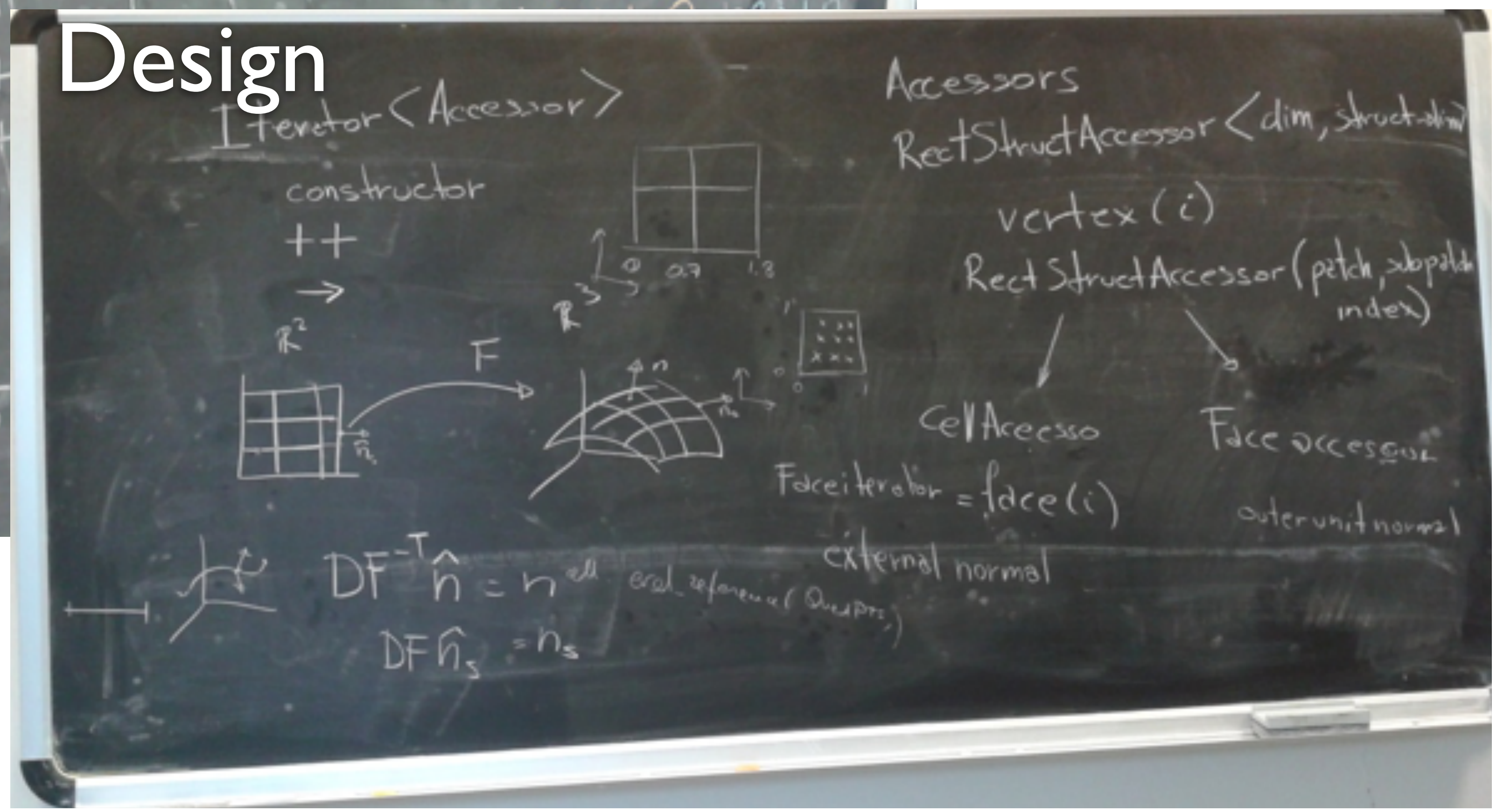
Conclusions



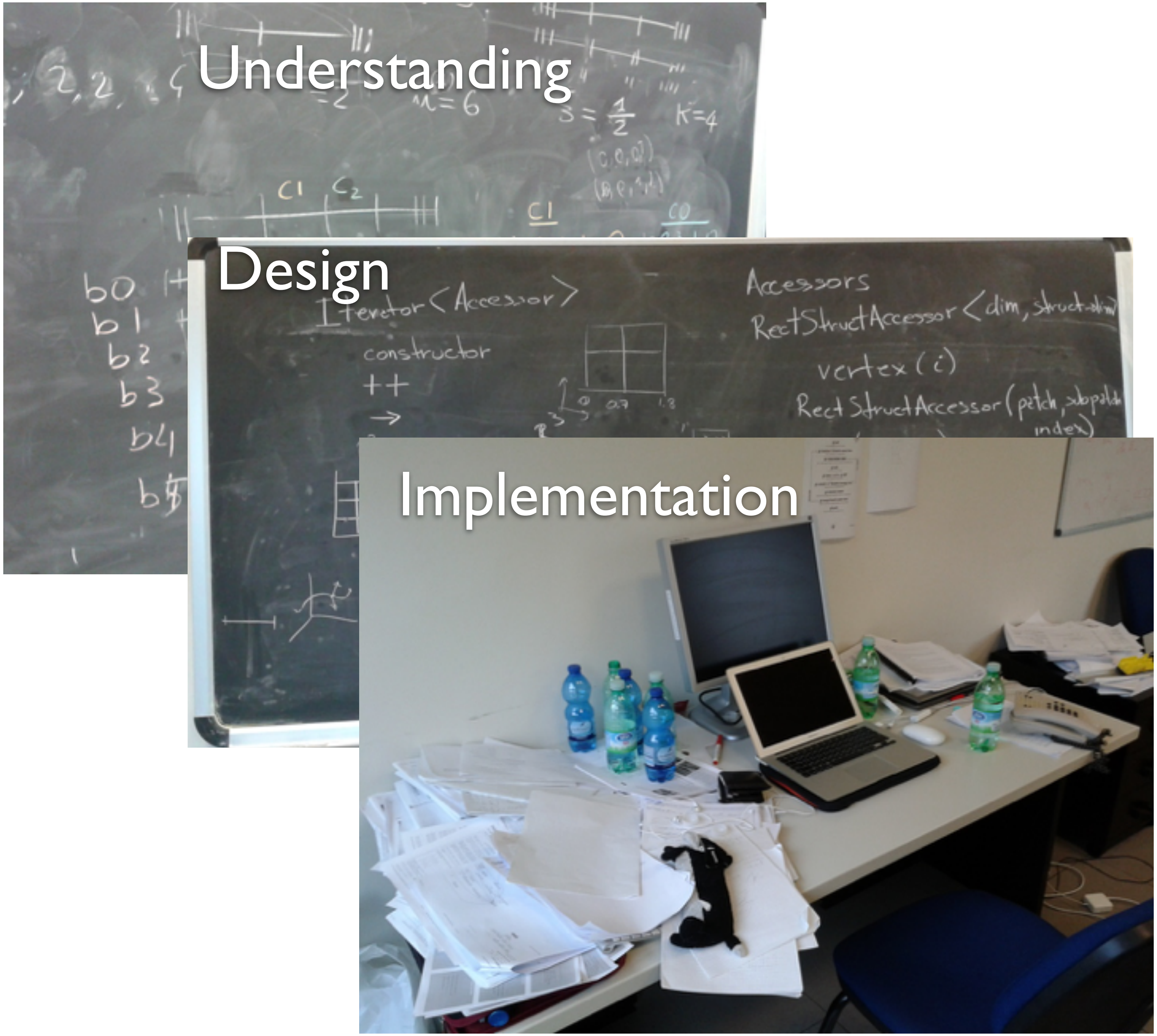
Understanding



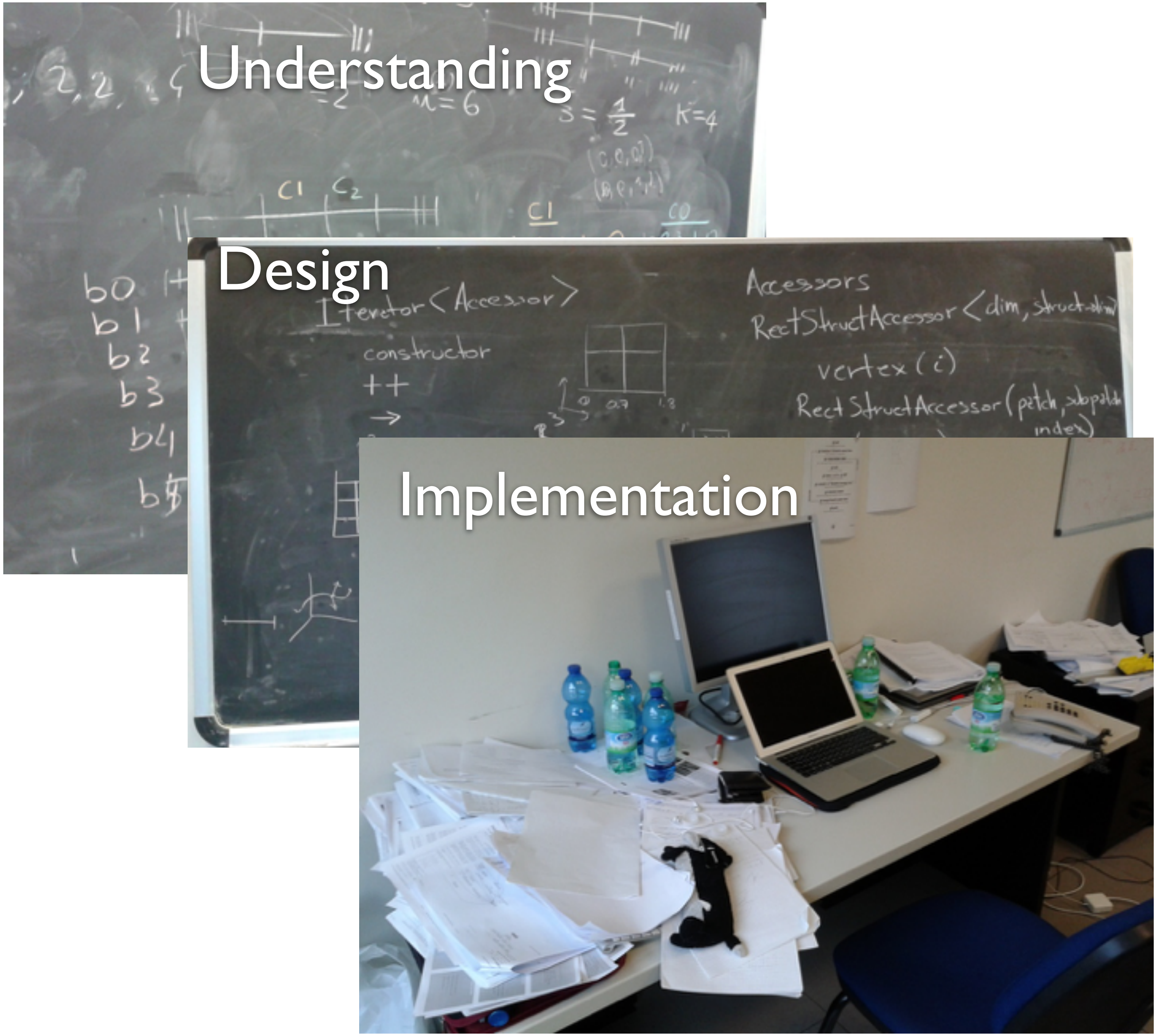

Understanding
s = 1/2
K=4
C1
C2
CO
b0
b1
b2
b3
b4


Design
Iterator<Accessor>
constructor
++
→
F
Accessors
RectStructAccessor<dim, struct-dim
vertex(i)
RectStructAccessor(patch, subpatch
index)
CellAccesso
Face accessor
Faceiterator = face(i)
outer unit normal
external normal

MHPC
Master in High Performance Computing

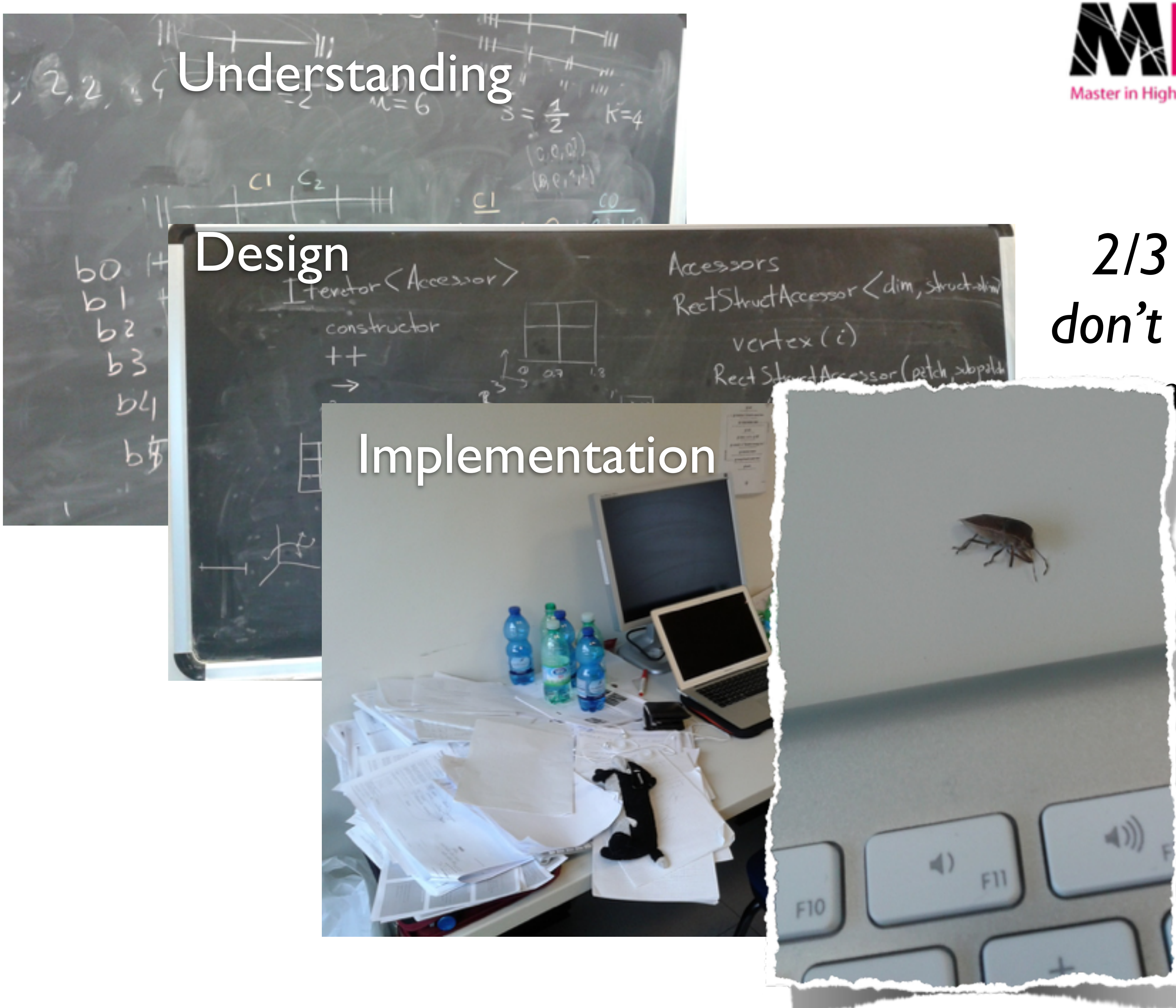


MHPC
Master in High Performance Computing

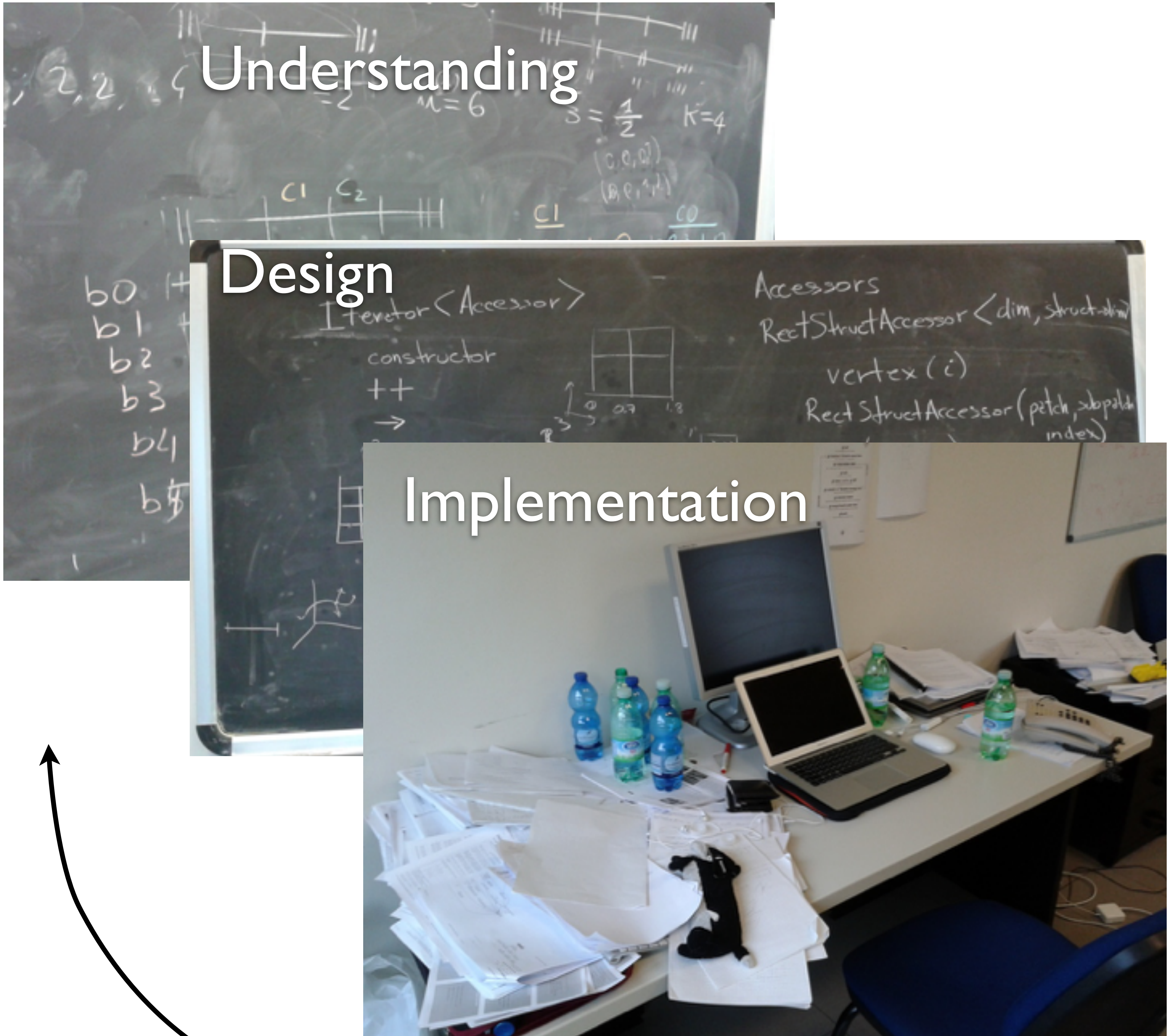


2/3 Phases don't require a computer.

MHPC
Master in High Performance Computing



2/3 Phases don't require a computer.



MHPC
Master in High Performance Computing

2/3 Phases don’t require a computer.

Enjoy!