UNIVERSITÀ
DI PARMA

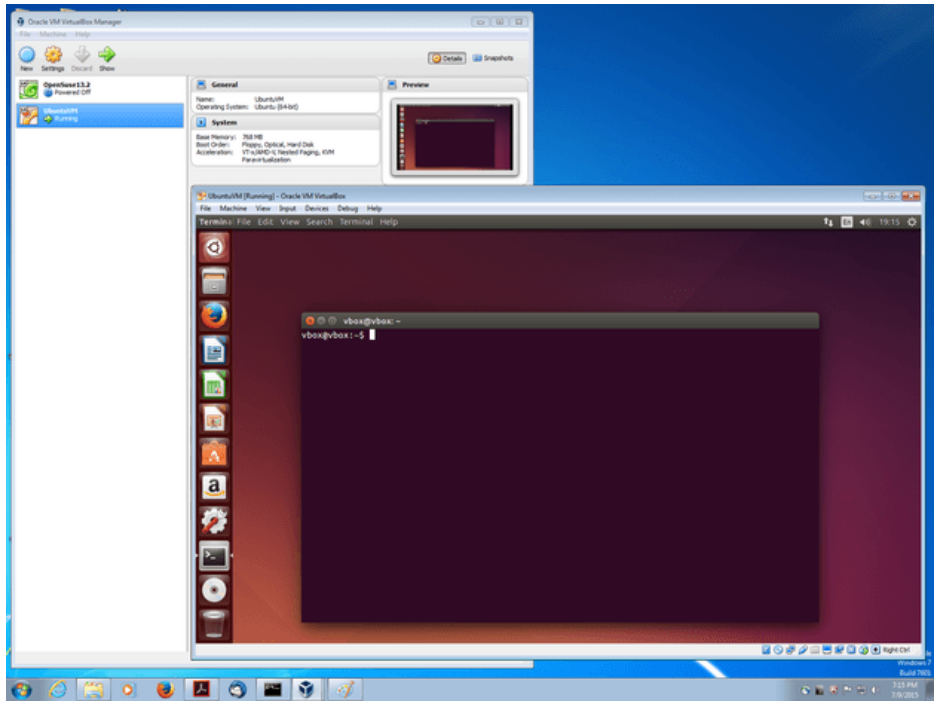
Informatica e Laboratorio di Programmazione

Python & C++

Alberto Ferrari

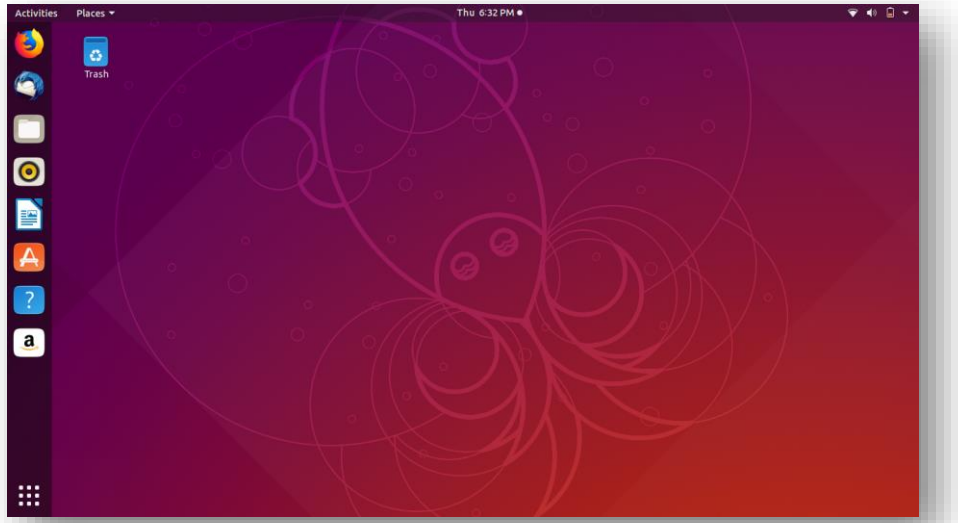
- ***virtualizzazione***: ***astrarre*** le componenti ***hardware*** (fisiche), degli elaboratori al fine di renderle disponibili al software in forma di ***risorsa virtuale***
  - è possibile ***installare sistemi operativi*** su hardware virtuale
  - l'insieme delle componenti hardware virtuali (hard disk, RAM, CPU, scheda di rete) prende il nome di ***macchina virtuale***
  - sulla macchina virtuale può essere installato un nuovo sistema operativo con le relative applicazioni

- Oracle VM ***VirtualBox*** è un software gratuito (open source) per l'esecuzione di macchine virtuali
- *https://www.virtualbox.org/*



OS virtualizzati

- mediante software di virtualizzazione è possibile installare sistemi operativi *'ospitati'*
- es. ***Ubuntu***
  - famiglia GNU/Linux
  - distribuito liberamente con licenza GNU GPL
- https://www.ubuntu-it.org/download



- https://cppyy.readthedocs.io
- cppyy is an automatic Python-C++ bindings generator, for calling C++ from Python and Python from C++

Python vs C++

esempio performance

funzione C++
calcolo approssimato integrale


```
double f(double x) {
    return x * x + x;
}

double cpp_integral(double a, double b, int n) {
    /**
     * Estimate the area beneath the curve f, between the
     * abscissas a and b; the region is approximated as n rectangles
     */
    auto total = 0.0;
    auto dx = (b - a) / n;
    for (auto i = 0; i < n; ++i) {
        total += dx * f(a + dx * i);
    }
    return total;
}
```

funzione Python
calcolo approssimato integrale


```
def py_f(x: float) -> float:
    return x * x + x

def py_integral(a: float, b: float, n: int) -> float:
    """
    Estimate the area beneath the curve py_f, between the
    abscissas a and b; the region is approximated as n rectangles
    """
    total = 0.0
    dx = (b - a) / n
    for i in range(n):
        total += dx * py_f(a + dx * i)
    return total
```

inclusione di funzioni C++ in Python (linux)


```
#cppyy is an automatic Python-C++ bindings generator
# for calling C++ from Python
import cppyy
cppyy.include("cpp_integral.cpp")        #C++ -> Python

from cppyy.gbl import cpp_integral       #C++ function
from py_integral import py_integral      #Python function
import time
def main():
    print('C++ start')
    t0 = time.time()
    i = cpp_integral(1, 10, 10_000_000)
    t = time.time() - t0
    print("C++ end   \t",f"val. {i:10.4f}\t time{t:10.4f}")

    print('Python start')
    t0 = time.time()
    i = py_integral(1, 10, 10_000_000)
    t = time.time() - t0
    print("Python end\t",f"val. {i:10.4f}\t time{t:10.4f}")
```

GUI Python & C++ objects

esempio game

C++ classes .h

```
class Actor
{
public:
    virtual void move() = 0;
    virtual void collide(Actor* other) = 0
;
    virtual vector<int> position() = 0;
    virtual vector<int> symbol() = 0;
    virtual ~Actor() {}
};
```

```
class Arena
{
private:
    int w_, h_;
    vector<Actor*> actors_;
public:
    Arena(int width, int height);
    void add(Actor* a);
    void remove(Actor* a);
    void move_all();
    bool check_collision(Actor* a1, Actor* a2);
    vector<Actor*> actors() { return actors_; }
    vector<int> size() { return {w_, h_}; }
    int width() { return w_; }
    int height() { return h_; }
    ~Arena();
};
```

C++ classes .h

```
class Ball : public Actor
{
private:
    Arena* arena_;
    int x_, y_, dx_, dy_;
    static const int W = 20, H = 20, SPEED = 5;
public:
    Ball(Arena* arena, int x, int y);
    void move();
    void collide(Actor* other);
    vector<int> position();
    vector<int> symbol();
};
```

```
class Turtle : public Actor
{
private:
    Arena* arena_;
    int x_, y_, dx_, dy_;
    static const int W = 20, H = 20, SPEED = 2;
public:
    Turtle(Arena* arena, int x, int y);
    void move();
    void collide(Actor* other);
    vector<int> position();
    vector<int> symbol();
    void go_left();
    void go_right();
    void go_up();
    void go_down();
    void stay();
};
```

C++ classes .cpp

```
…
void Arena::move_all() {
    auto acts = actors();
    reverse(begin(acts), end(acts));
    for (auto a : acts) {
        auto prev = a->position();
        a->move();
        auto curr = a->position();
        if (curr != prev) {
            for (auto other : acts) {
                if (other != a && check_collision(a, other)) {
                    a->collide(other);
                    other->collide(a);
                }
            }
        }
    }
}
…
```

C++ classes .cpp

```
...
void Ball::collide(Actor* other) {
    auto ghost = dynamic_cast<Ghost*>(other);
    if (ghost == nullptr) {
        auto op = other->position();
        if (op[0] < x_) {
            dx_ = SPEED;
        } else {
            dx_ = -SPEED;
        }
        if (op[1] < y_) {
            dy_ = SPEED;
        } else {
            dy_ = -SPEED;
        }
    }
}

...
```

```
import cppyy
cppyy.include("actor.cpp")
cppyy.include("bounce.cpp")
from cppyy.gbl import Arena, Ball, Ghost, Turtle

import g2d
arena = Arena(320, 240)
b1 = Ball(arena, 40, 80)
b2 = Ball(arena, 80, 40)
g = Ghost(arena, 120, 80)
turtle = Turtle(arena, 80, 80)
sprites = g2d.load_image("sprites.png")

def update():
    arena.move_all()  # Game logic

    g2d.fill_canvas((255, 255, 255))
    for a in arena.actors():
        g2d.draw_image_clip(sprites, a.position(), a.symbol())

...
```