



# C++ Python binding

---

- **It is cool**
- **Effective reuse of existing code/projects**
- **To fulfill different requirements**
  - **execution speed**
  - **development speed**
  - **easy on-site customization**



# Examples

---

Python

SQLite

Berkeley DB

wxPython

PyKDE

PyQT

cctbx

JPype

PyGTK

TnFOX

Python for .NET

Civilization IV

PyWin32

Pygame (SDL)



# C++ Python binding tools

---

- **SWIG**
- **SIP**
- **PyCXX**
- **Robin**
- **Boost.Python + pyplusplus**



# Boost.Python features list

---

- **Extending and embedding**
- **Classes**
- **Functions**
- **C++ to Python exception translation**
- **Iterators**
- **STD containers**



## Example - class “world”

---

```
struct world{  
    world(const std::string& msg)  
    : m_msg(msg)  
    {}  
    void set(const std::string& msg)  
    { m_msg = msg; }  
  
    const std::string& greet() const  
    { return m_msg; }  
private:  
    std::string m_msg;  
};
```



# Exposed class “world”

---

```
using namespace boost::python;
```

```
BOOST_PYTHON_MODULE(hello){  
  class_<world>("world", init<const std::string&>())  
    .def("greet"  
        , &world::greet  
        , return_value_policy<copy_const_reference>())  
    .def("set", &world::set);  
}
```



# Python session

---

```
>>> import hello
```

```
>>> w = hello.world( "Good morning" )
```

```
>>> w.greet()
```

```
'Good morning'
```



# Automatically exposed class “world”

---

```
using namespace boost::python;
```

```
BOOST_PYTHON_MODULE(pyplusplus) {
```

```
    class_< world >( "world"
        , init< std::string const & >(( arg("msg") ))
        /*[ undefined call policies ]*/ )
        .def( "set"
            , &world::set
            , ( arg("msg") )
            , default_call_policies() )
        .def( "greet"
            , &world::greet
            , return_value_policy
              <copy_const_reference, default_call_policies>());
}
```





# pyplusplus introduction

---

## pyplusplus:

is an object-oriented framework for creating a code generator for Boost.Python library

## Goals:

- ✓ developer will be able to apply the changes before code is generated
- ✓ developer will be able to work on whole project\library at once



# pyplusplus features list

---

- **C++ parser independence**
- **Small and intuitive GUI, that includes wizard**
- **Ability to add\remove\modify code anywhere**
- **Scalability, works well on small and big projects**
- **Short learning curve**



# pyplusplus design

---

- **Code creators package**
- **Module creator package**
- **File writers package**
- **Code repository package**



# pyplusplus development process

---

- **Main guideline: highest quality of generated code**
- **Agile software development methodology**
- **Real world projects used in testing process**
  - **boost.date\_time**
  - **EasyBMP**
  - **Qt.Xml**
  - **TnFOX**



# Benefits

---

- **Extending and embedding**
- **Readability counts**
- **Interface definition language is actually C++**
- **Boost.Python enables us to think hybrid**
- **Component\package based development**
- **Get first working version in few hours**



# Pitfalls

---

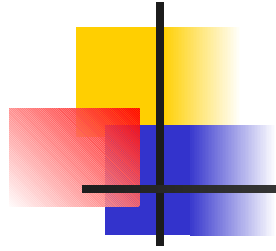
- **It takes time to compile code exposed with Boost.Python library**
- **pyplusplus does not have good documentation**



# Conclusion

---

- **Code generators is a valuable technique**
- **Multi-language development actually works**



**Thank you**

**Questions?**