# PyBind Documentation

## Header and namespace conventions

Alla kod-examples kommer inkludera denna kod:



## Creating bindings for a simple function

Vi börjar med att skapa en python binding för en simple funktion *add()*.



*PYBIND11\_EMBEDDED\_MODULE()* skapar en modul som python kan importera, i detta fall heter modulen *lab.* Sedan används m som ett interface för att skapa bindings. I detta fall använder jag *m.def()* för att binda funktionen *add()* till *lab* modulen.

Inom *m.def()* kan du se tre olika argument. Den första *”add”* bestämmer vad funktionen skall heta för python. Den andra, *&add,* är en referens till funktionen som ska bindas. Den tredje, *”Adds a and b.”* är en kommentar till funktionen som kommer visas om du kör *help(add)* i python. Endast de första två är obligatoriska, och fler kommer läggas till i framtida funktioner.

För att använda funktionerna i python:



## Keyword arguments

Ett nytt argument för *m.def()* är *py::arg(),* vilket används i följande sätt.



Detta gör det möjligt att specificera vilket argument man ändrar i funktionen, och dessutom låter dig lägga till default argument. För att lägga till default argument i sin bound funktion gör man följande:



Nu kommer *add()* returnera 3 om inga argument läggs till.

## Exporting Variables

För att exportera variabler så att dem är tillgängliga för python gör man följande.



Där *”coolNumber”* är namnet som variabeln ska ha i python, och *coolNumber* är vilken variabel som den tar ifrån. Notera att det går att ändra variabeln i python, men den kommer inte ändras i C++!

## Custom Types / Classes

Följande är ett exempel på hur en klass bindas.



Där *py::class\_<>()* skapar en klass struktur i python. *<Pet>* bestämmer vilken klass som används, *m* är vilken interface som används i klassen, och *”Pet”* är namnet på klassen i python. Sedan defineras alla funktioner i klassen ungefär på samma sätt som förut. *py::init<>()* är en funktion som säger att konstruktorn ska ta en i detta fall *<const std::string &>* som argument.

## Inheritance and automatic upcasting

Att lägga till inheritance med pybind är lätt.



Skapa klassen som vanligt, men när du bestämmer vilken klass den ska komma ifrån *<Pet>,* skriv också vilken klass som den är inherited från *<Dog, Pet>*. Annars är resten som en vanlig klass.

## Overloaded methods

Det är möjligt att göra flera funktioner med samma namn men olika funktionalitet baserat på argumentens typ, men att få dem att fungera i PyBind är lite weird.



Skriv funktionen som vanligt, men funktionen måste castas till… Vad detta än nu är. Det följer i alla fall denna struktur.



Fråga mig inte varför det fungerar, men det fungerar.

## Extra stuff

Tänk på att funktionerna som skickas över körs i C++, så man har access till allt som Python kallar på detta sätt, man kan exempelvis göra så här för att skicka information till C++ från Python.



Behöver du mer information finns detta på denna hemsida.

<https://pybind11.readthedocs.io/en/stable/basics.html>