Parsing Python Code (0)

```
>>> import ast
>>> code = "a=1"
>>> tree = ast.parse(code)  # turn the code into an abstract syntax tree
>>> print ast.dump(tree)  # returns a sformatted representation of the tree
Module(body=[Assign(targets=[Name(id='a', ctx=Store())], value=Num(n=1))])
```

Parsing Python Code (1)

```
>>> tree = ast.parse("def fib(n): return n if n< 2 else fib(n-1) + fib(n-2)")
>>> print ast.dump(tree)
```

Pythran Pass Manager (0)

```
>>> from pythran import passmanager
>>> pm = passmanager.PassManager("tutorial_module")
```

Pythran Pass Manager (1)

```
>>> [x for x in dir(pm) if not x.startswith('__')]
['apply', 'dump', 'gather', 'module_name']
```

Pythran Backends (0)

```
>>> from pythran import backend
>>> cxx = pm.dump(backend.Cxx, tree)
>>> "\n".join("\n".join(s.generate()) for s in cxx)
'#include <pythran/pythran.h>\n#include <pythran/pythran_gmp.h>\nnamespace __tut
orial_module\n{\n  ;\n  struct fib\n  {\n    typedef void callable;\n    templat
e <typename argument_type0 >\n    struct type\n    {\n      typedef typename ass
ignable<typename std::remove_cv<typename std::remove_reference<argument_type0>::
```

```
type>::type>::type result_type;\n    }  \n    ;\n    template <typename argument
_type0 >\n    typename type<argument_type0>::result_type operator()(argument_typ
e0 const & n) const\n    ;\n  }  ;\n  template <typename argument_type0 >\n  typ
ename fib::type<argument_type0>::result_type fib::operator()(argument_type0 cons
t & n) const\n  {\n    return ((n < 2L) ? n : (fib()((n - 1L)) + fib()((n - 2L))
));\n  }\n}'
```

Pythran Backends (1)

```
>>> py = pm.dump(backend.Python, tree)
>>> print py
def fib(n):
    return (n if (n < 2) else (fib((n - 1)) + fib((n - 2))))
```

Passes (0)

```
>>> from pythran import passes
>>> tree = ast.parse("a,b = 1,3.5")
>>> _ = pm.apply(passes.NormalizeTuples, tree)  # modification is done in-place
>>> print pm.dump(backend.Python, tree)
if 1:
    __tuple10 = (1, 3.5)
    a = __tuple10[0]
    b = __tuple10[1]
```

Passes (1)

```
>>> tree = ast.parse('namespace_ = new = 1\nnamespace = namespace_ + new')
>>> _ = pm.apply(passes.NormalizeIdentifiers, tree)  # output is a renaming dict
ionnary
>>> print pm.dump(backend.Python, tree)
namespace_ = new_ = 1
namespace__ = (namespace_ + new_)
```