PYREDS

INTERACTIVE COMMAND-LINE INTERPRETER

Jean-Pierre Miceli

AGENDA

- Pyreds overview
  - What is pyreds
  - What is NOT pyreds
- pyreds features
- pyreds *Module* creation

PYREDS OVERVIEW

WHAT IS PYREDS

- Pyreds in an **interactive command-line interpreter**
  - Simplifies and accelerates activities such as debug and testing
- pyreds is **Generic** - project-agnostic - tool
  - It is fully extendable though plugs-in, called *Modules*

WHAT IS NOT PYREDS

- ➤ It is NOT specific to a project
  - It is meant to be reused across projects

- ➤ It does not solve a specific engineering task
  - It enables implementing *Modules* for specific tasks
  - It enables using such *Modules* efficiently

PYREDS FEATURES

FEATURES

- **Scripting** support
- Full support for a modern and easy-to-use programming language, **Python**
- **Tab completion** and **command history** support to ease interactive sessions
- For *Modules* (plug-in) developers
  - Simple interface for *Modules* creation
  - Access to full power of Python !

PYREDS FEATURES (2)

- Documentation enabled within components
- Multi-platforms (thanks to Python)
  - N.B: *Modules* may not be multi-platforms

SUPPORTED COMMAND TYPES

- Sub-commands

```
demo.subCommand
```

- Indexed commands (not implemented yet)

```
demo.indexdCommand[‘index’]
```

- Functions

```
demo.function(‘param1’, ‘param2’)
```

- Possible to mix them

```
demo.subCmd.idxCmd[idx].otherSubCmd.func(‘param’)
```

- Function is the end of a command (nothing after a function)

COMPONENT CREATION

COMPONENTS CREATION

- Full Python code !
- Component: class which inherits form pyr*eds.Module*
- Commands: class which inherits form *pyreds.Command*
- Use of a small set of keyword to implement wanted features
  - Function: method called *_function(), _input* lists
- Usage: simple instantiation of the commands