Pythonect Documentation

Release 0.6.0

Itzik Kotler

July 22, 2013

CONTENTS

Welcome to Pythonect's documentation. This documentation is divided into different parts. I recommend that you get started with *Installation* and then head over to the *Tutorial*. If you'd rather dive into the internals of Pythonect, check out the *API Reference* documentation.

Pythonect is dependent on one external library: the NetworkX graph library. This library is not documented here. If you want to dive into its documentation, check out the following link: NetworkX Documentation

---

Note: This is the main documentation for the Pythonect project. The contents of this site are automatically generated via Sphinx based on the Python docstrings throughout the code and the reStructuredText documents in the doc/ directory of the git repository. If you find an error in the documentation, please report it in the bug tracker here, or even better, submit a pull request!

CHAPTER

ONE

USER'S GUIDE

This part of the documentation, which is mostly prose, begins with some background information about Pythonect, then focuses on step-by-step instructions for building applications with Pythonect.

1.1 Foreword

Read this before you get started with Pythonect. This hopefully answers some questions about the purpose and goals of the project, and when you should or should not be using it.

1.1.1 What is Pythonect?

Pythonect is a new, experimental, general-purpose dataflow programming language based on Python. It provides both a visual programming language and a text-based scripting language. The text-based scripting language aims to combine the quick and intuitive feel of shell scripting, with the power of Python. The visual programming language is based on the idea of a diagram with "boxes and arrows".

The Pythonect interpreter (and reference implementation) is a free and open source software written completely in Python, and is available under the BSD license.

1.1.2 Why Pythonect?

Pythonect, being a dataflow programming language, treats data as something that originates from a source, flows through a number of processing components, and arrives at some final destination. As such, it is most suitable for creating applications that are themselves focused on the "flow" of data. Perhaps the most readily available example of a dataflow-oriented applications comes from the realm of real-time signal processing, e.g. a video signal processor which perhaps starts with a video input, modifies it through a number of processing components (video filters), and finally outputs it to a video display.

As with video, other domain problems (e.g. image processing, data analysis, rapid prototyping, and etc.) can be expressed as a network of different components that are connected by a number of communication channels. The benefits, and perhaps the greatest incentives, of expressing a domain problem this way is scalability and parallelism. The different components in the network can be maneuvered to create entirely unique dataflows without necessarily requiring the relationship to be hardcoded. Also, the design and concept of components make it easier to run on distributed systems and parallel processors.

Continue to *Installation* or *Tutorial*

1.2 Installation

Pythonect works with Python version 2.6 and greater, but it will not work (yet) with Python 3. Dependencies are listed in `setup.py` and will be installed automatically as part of any of the techniques listed below.

1.2.1 Using pip or easy_install

Easiest way to install Pythonect is to use `pip`:

```
pip install Pythonect
```

And second to easiest is with `easy_install`:

```
easy_install Pythonect
```

Note: Using easy_install is discouraged. Why? Read here.

1.2.2 Using git repository

Regular development happens at our GitHub repository. Grabbing the cutting edge version might give you some extra features or fix some newly discovered bugs. We recommend not installing from the git repo unless you are actively developing *Pythonect*. To clone the git repository and install it locally:

```
git clone git://github.com/ikotler/pythonect.git
cd pythonect
python setup.py install
```

Alternatively, if you use pip, you can install directly from the git repository:

```
pip install \
        git+git://github.com/ikotler/pythonect.git@master#egg=pythonect \
        -r https://github.com/ikotler/pythonect/raw/master/doc/requirements.txt
```

1.2.3 Using archives (tarball)

Visit our PyPI Page to grab the archives of both current and previous stable releases. After untarring, simply run `python setup.py install` to install it.

1.3 Tutorial

This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of Pythonect's most noteworthy features, and will give you a good idea of the language's flavor and style.

Pythonect is based on Python and uses many of its features. If you are unfamiliar with Python, start with this Python Tutorial.

1.3.1 Using the Pythonect Interpreter

Pythonect provides an interpreter named `pythonect` for evaluating Pythonect programs and expressions interactively.

Invoking the Interpreter

Using the Pythonect interpreter is quite easy. Once the interpreter is *installed*, you're ready to go. You can invoke the interpreter from the command line as follows:

```
$ pythonect
```

The interpreter prints its sign-on message and leaves you at its >>> prompt:

```
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[Pythonect 0.5.0.dev12] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

In this interactive mode, you can evaluate arbitrary Pythonect expressions:

```
>>> 1+1
2
>>> "Hello, world" -> print
<MainProcess:MainThread> : Hello, world
```

To exit the interpreter, just type the `quit` command at the beginning of a line (on Unix systems, typing the end-of-file character `Control-D` will do the same).

Of course you can also run your scripts directly from the command line, as follows:

```
$ pythonect myscript.p2y
```

This executes the script in batch mode. Add the *-i* option if you prefer to run the script in interactive mode:

```
$ pythonect -i myscript.p2y
```

A number of other command line options are available; try pythonect *-h* for a list of those.

The Interpreter and Its Environment

Executable Pythonect Scripts

On BSD'ish Unix systems, Pythonect scripts can be made directly executable, like shell scripts, by putting the line:

```
#! /usr/bin/env pythonect
```

(assuming that the interpreter is on the user's `PATH`) at the beginning of the script and giving the file an executable mode. The `#!` must be the first two characters of the file.

The script can be given an executable mode, or permission, using the **chmod** command:

```
$ chmod +x myscript.p2y
```

It can also accept arguments from the command line (these are available in Pythonect by accessing `sys.argv`), as follows:

```
$ cat calc.p2y
#! /usr/bin/env pythonect
int(sys.argv[1]) + int(sys.argv[2]) -> print

$ pythonect calc.p2y 1 2
<MainProcess:MainThread> : 3
```

The `.pythonect_history` File

When running interactively, the Pythonect interpreter usually employs the GNU readline library to provide some useful command line editing facilities, as well as to save command history. The cursor up and down keys can then be used to walk through the command history, existing commands can be edited and resubmitted with the `Enter` key, etc. The command history is saved in the `.pythonect_history` file in your home directory between different invocations of the interpreter.

1.3.2 Hello World Program

Pythonect provides both a visual programming language and a text-based scripting language.

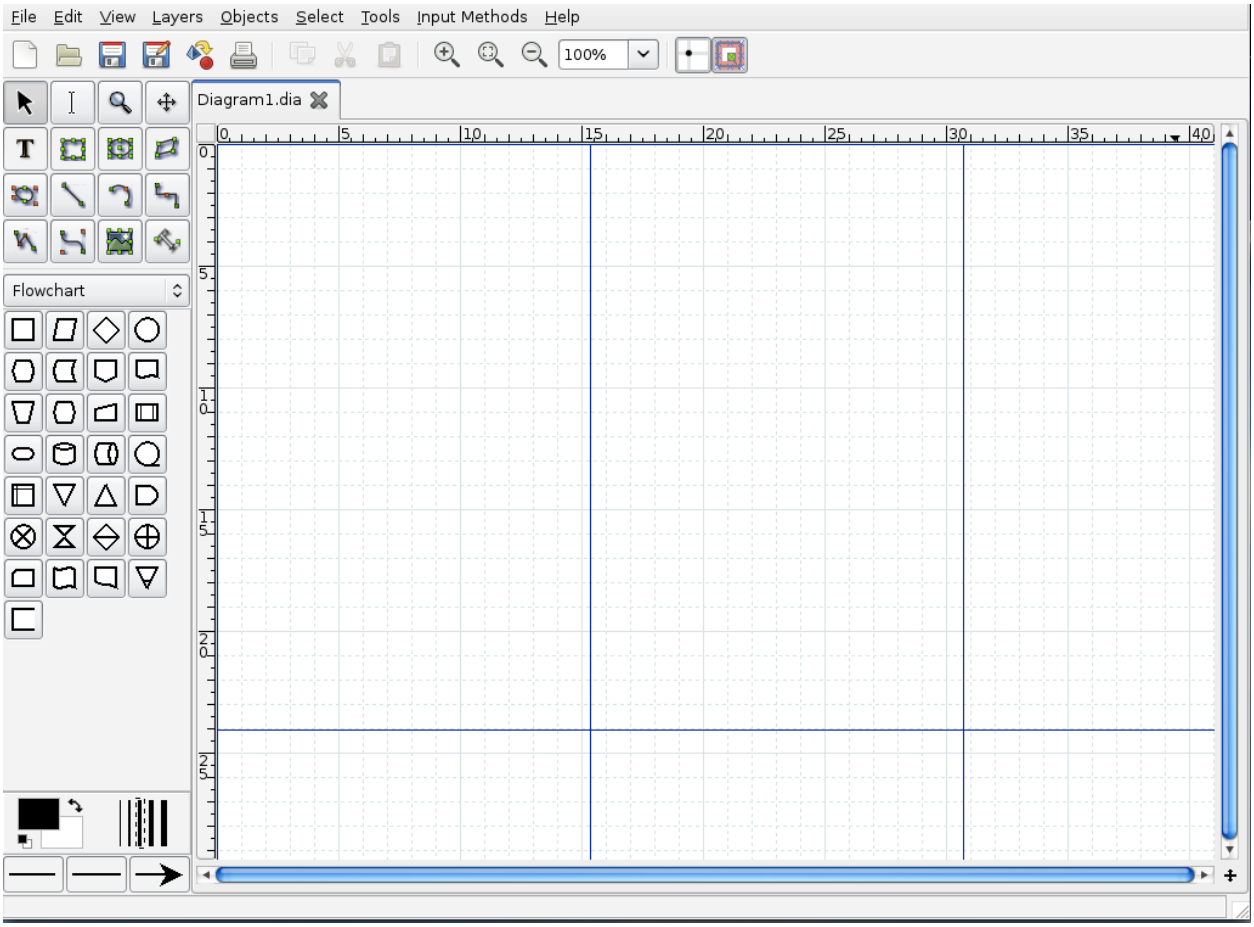
The following is an example Hello world program in both visual and text-based languages. Both versions consist of the same flow.

Visual Programming Version

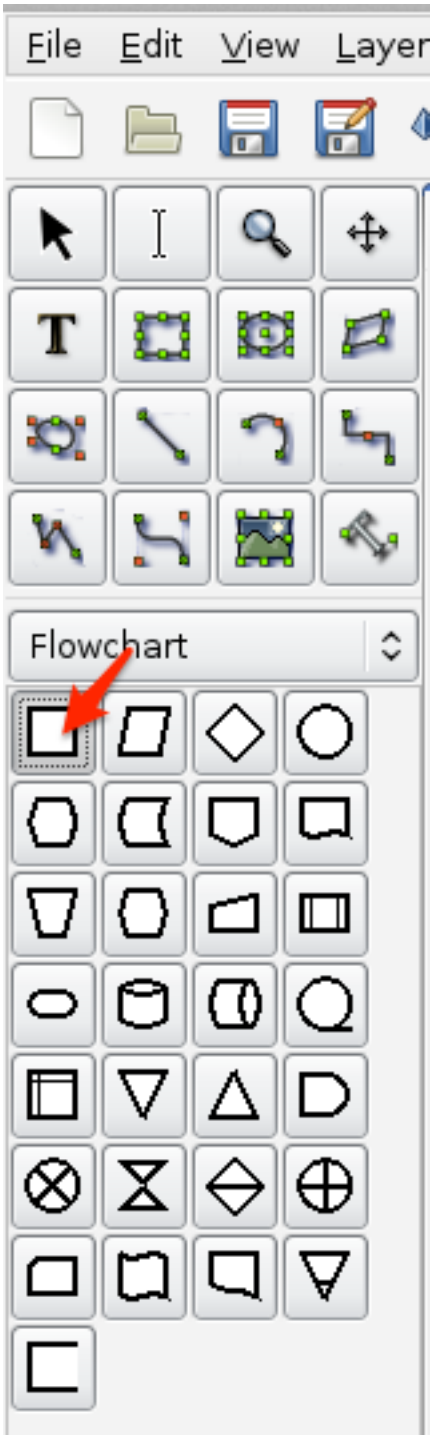
Pythonect supports multiple graph describing languages and diagram formats. For the Hello world example I will be using Dia.

Dia is a free and open source general-purpose diagramming software. For instructions on how to download and install **dia**, please visit Dia Website

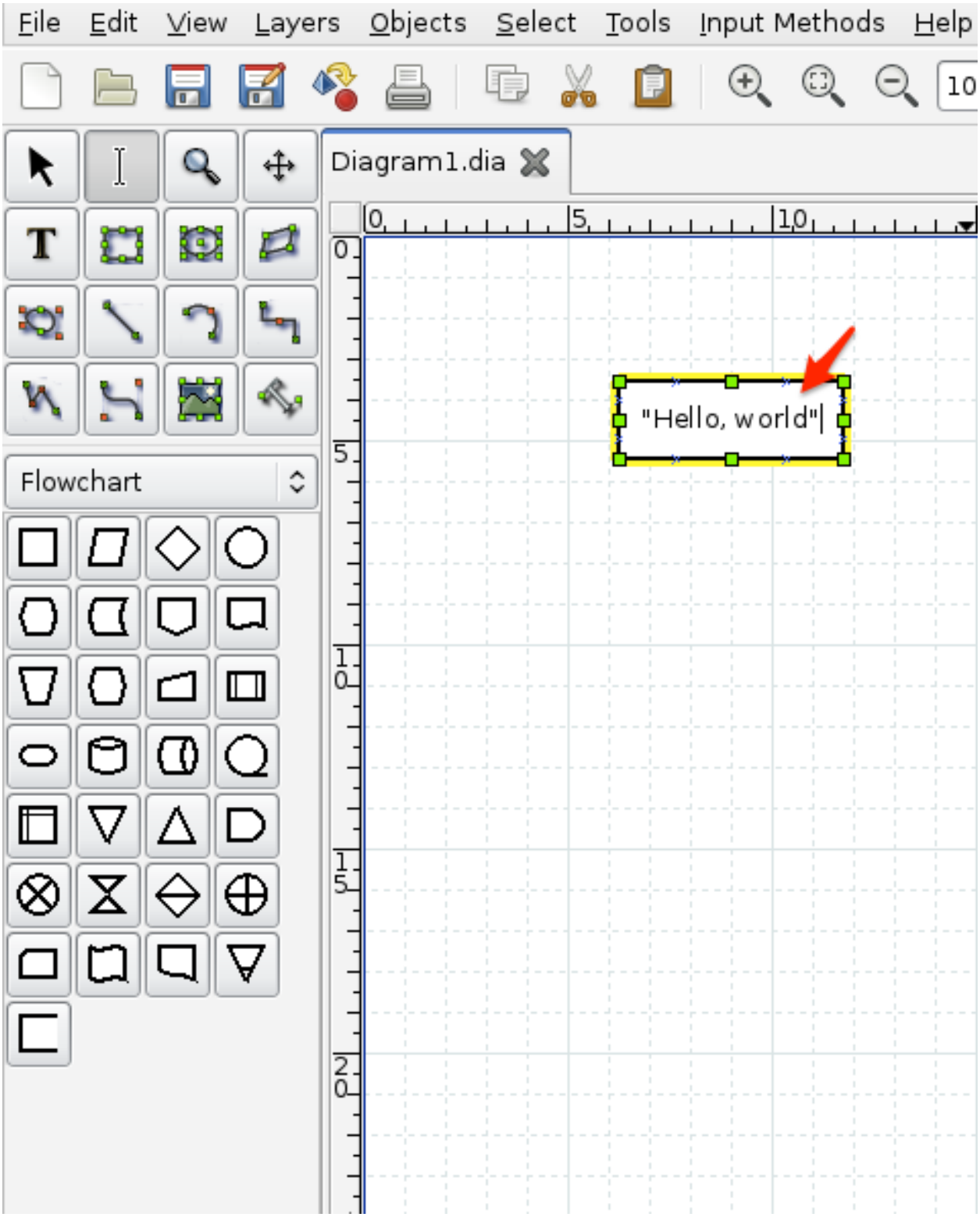
When you launch **dia** you should see an empty `Diagram1.dia` tab, like this:



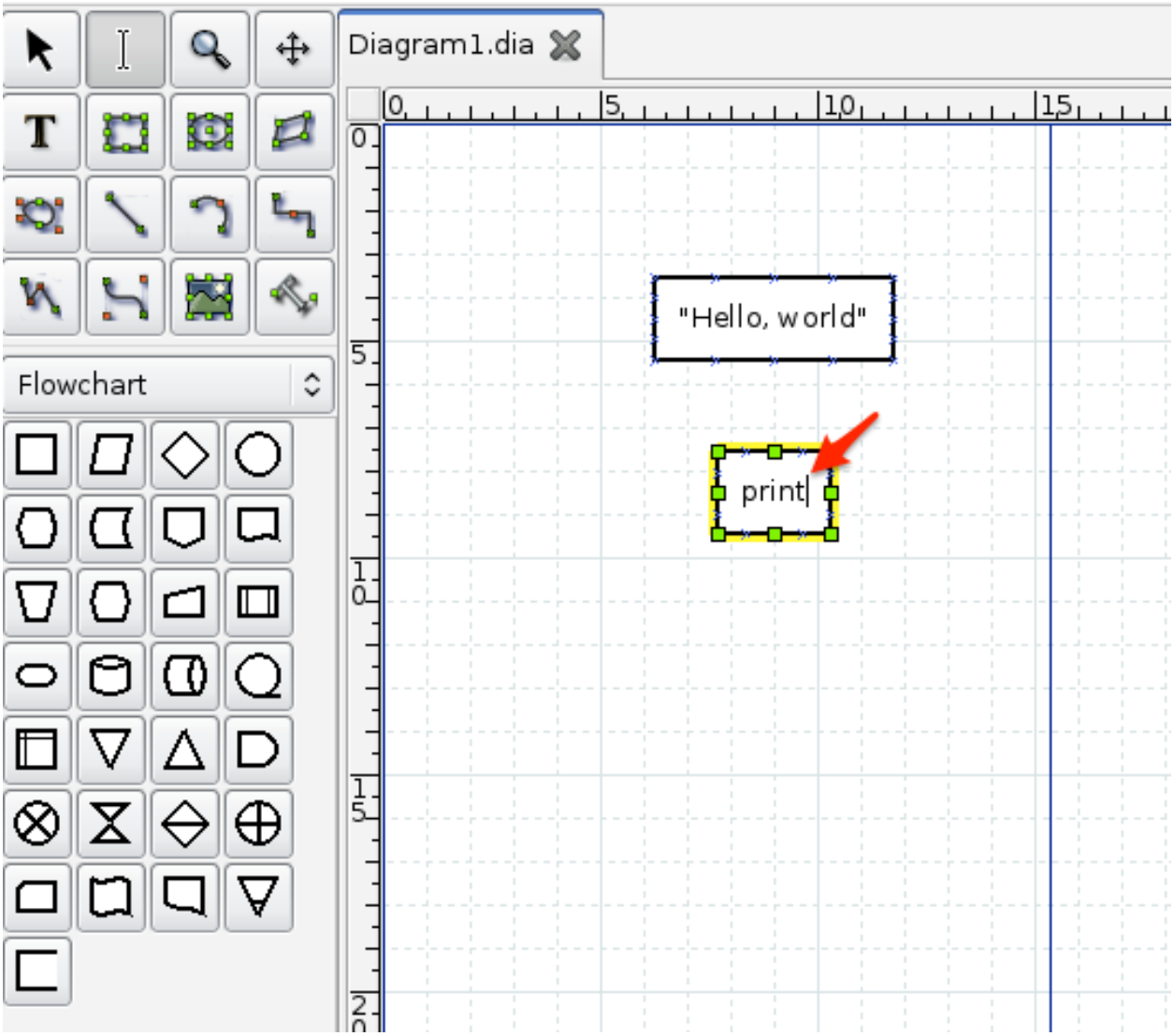
Start by clicking on the box (aka. Process/Auxliary Operation) shape to select it:



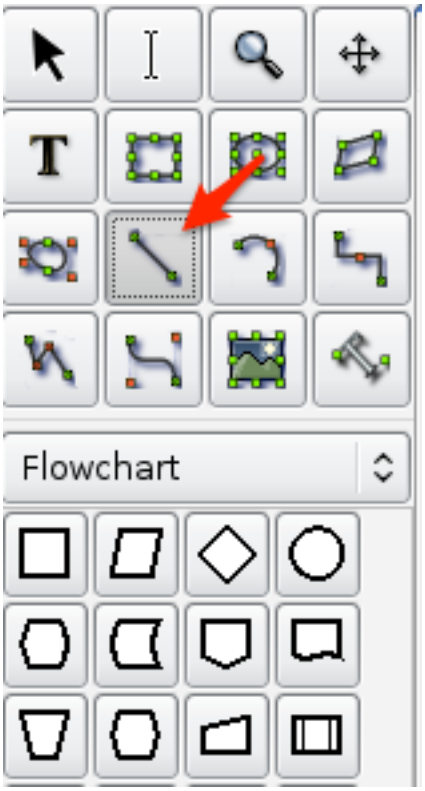
Now, click anywhere on the grid, a box (aka. Process/Auxliary Operation) should appear. Enter the text `"Hello, world"` (with quotes) in it:



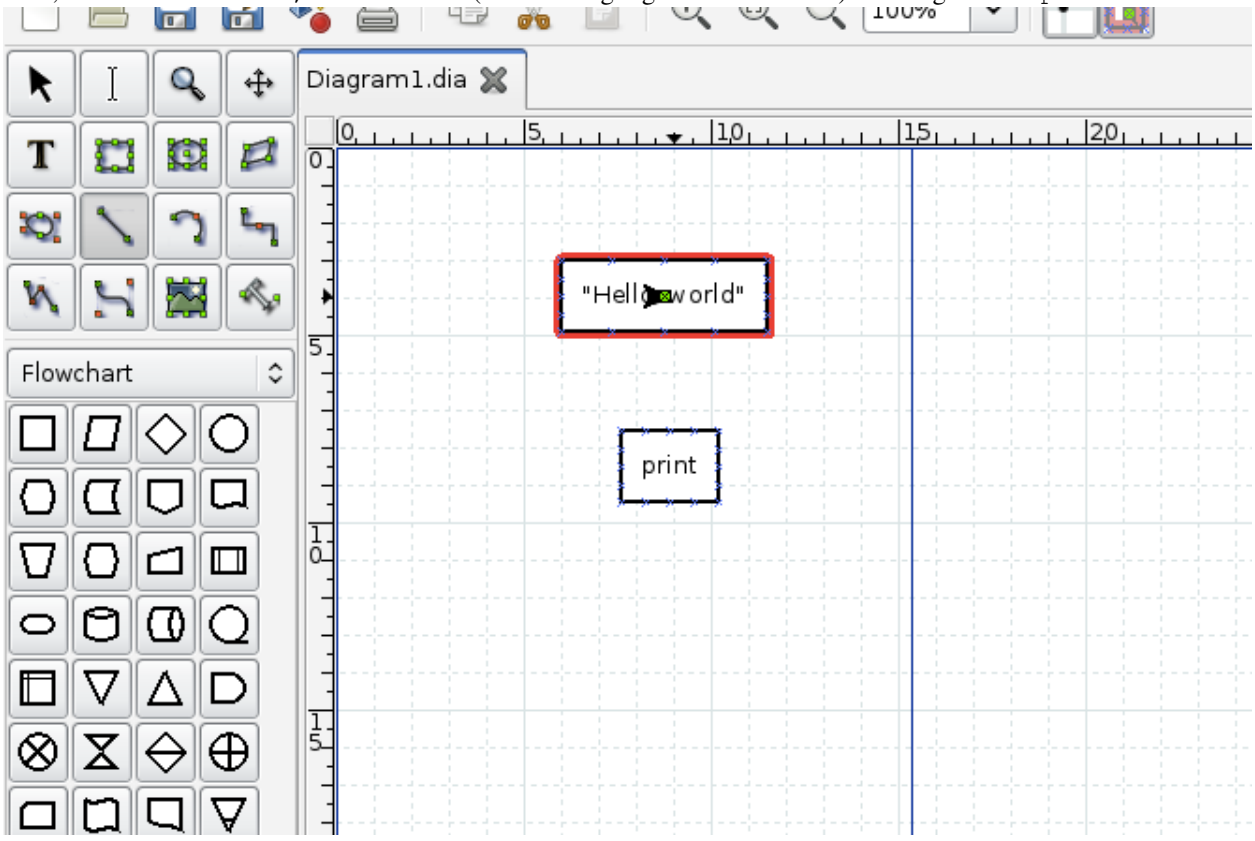
Next, click again on the grid, another box should appear. Enter the text `print` (without quotes) in it:



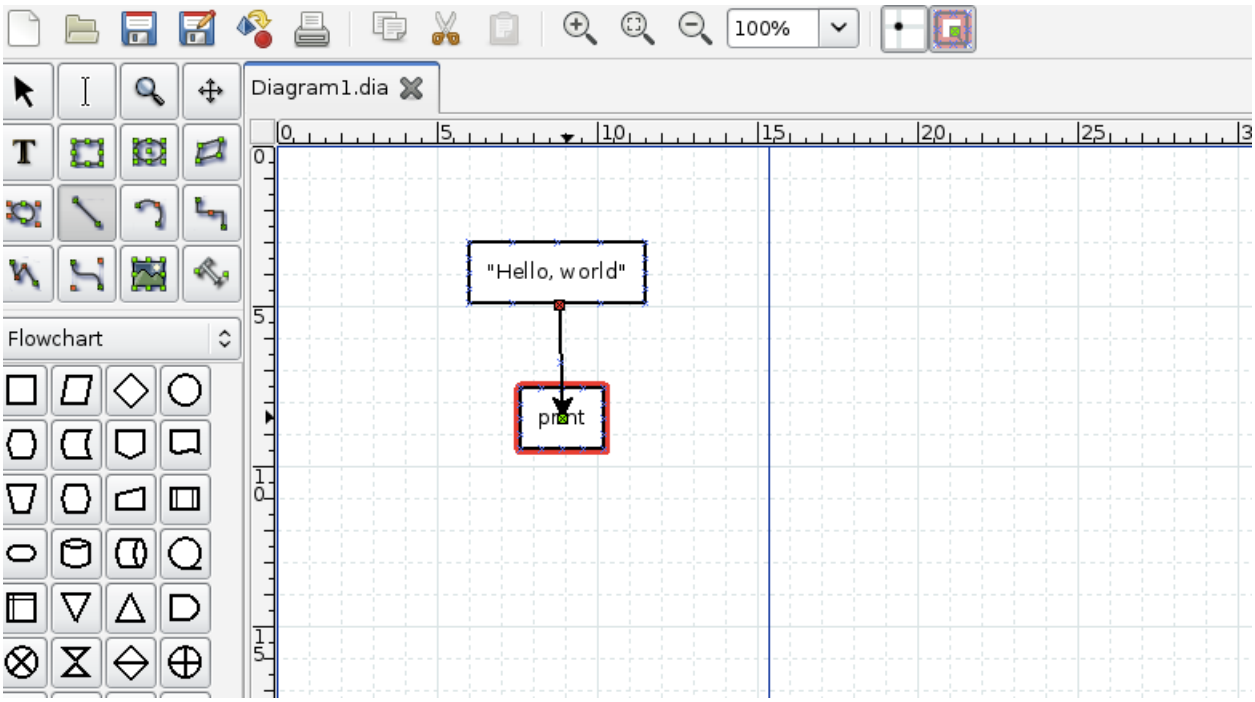
Let's connect the two boxes together. Start by clicking on the Line image to select it:



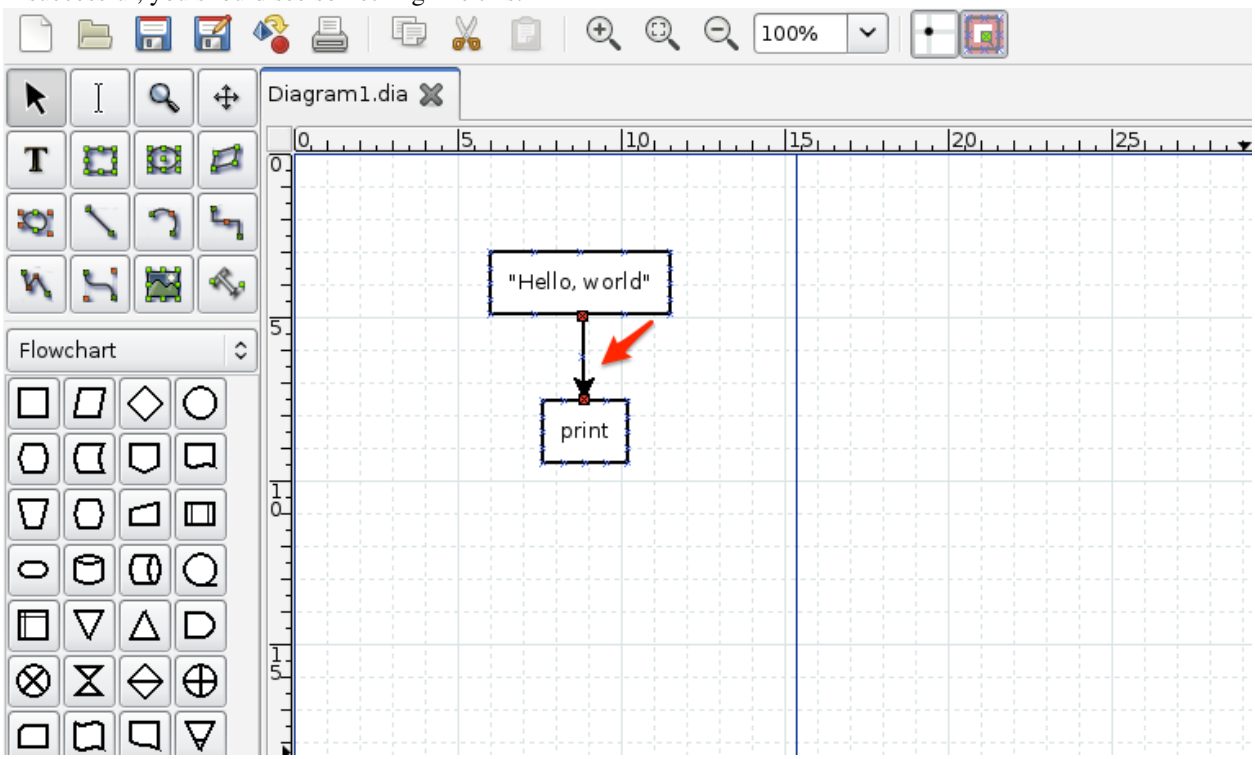
Now, click on the `"Hello, world"` box (it should highlight the box in Red) and drag it to the `print` box:



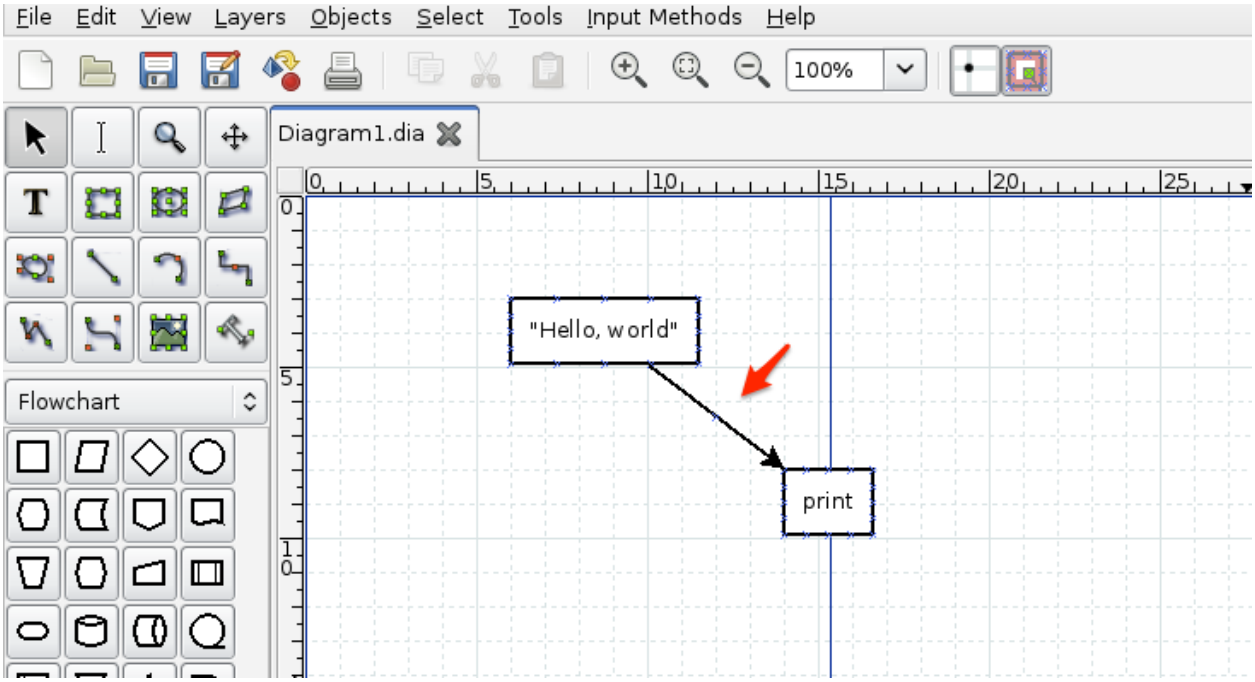
Like this:



If successful, you should see something like this:



To test that it's indeed connect, try dragging one of the boxes and see that the Line is following, like this:



If indeed the Line is following, then it's time to save the diagram. Let's save it as `HelloWorld.dia` and run it, as follows:

```
$ pythonect HelloWorld.dia
```

The output should be:

```
<MainProcess:MainThread> : Hello, world
```

Text-based Programming Version

Open your favorite editor and type:

```
"Hello, world" -> print
```

Save it as `HelloWorld.p2y` and run it as follows:

```
$ pythonect HelloWorld.p2y
```

The output should be:

```
<MainProcess:MainThread> : Hello, world
```

To break it down: `"Hello, world"` is a literal String, `print` is a Python function, and `->` is a text-based data flow operator.

You can learn more about Pythonect's Data Flow Operators at the *Data Flow* section.

1.3.3 Data Flow

Depending on the scripting interface, directing the flow can be represented by plain text or interconnected lines

Text-based Symbols

The text-based scripting language aims to combine the quick and intuitive feel of shell scripting. Adopting a *Unix Pipeline*-like syntax.

`|` as Synchronous Forward

This operator pushs data to next operation and **do** block/wait for it finish. For example:

```
[1,2,3,4] | print
```

Will always print (each item in it's own thread and in this order): `1`, `2`, `3`, `4`.

`->` as Asynchronous Forward

This operator pushs data to the next operation and **do not** block/wait for it to finish. For example:

```
[1,2,3,4] -> print
```

May print (each item in it's own thread): `4`, `3`, `2`, `1` or `2`, `1`, `3`, `4` or even `1`, `2`, `3`, `4`.

"Boxes and Arrows"

Currently, in the Visual Programming Language, all lines are treated as asynchronous forward.

1.3.4 Variables

Assignment Statement

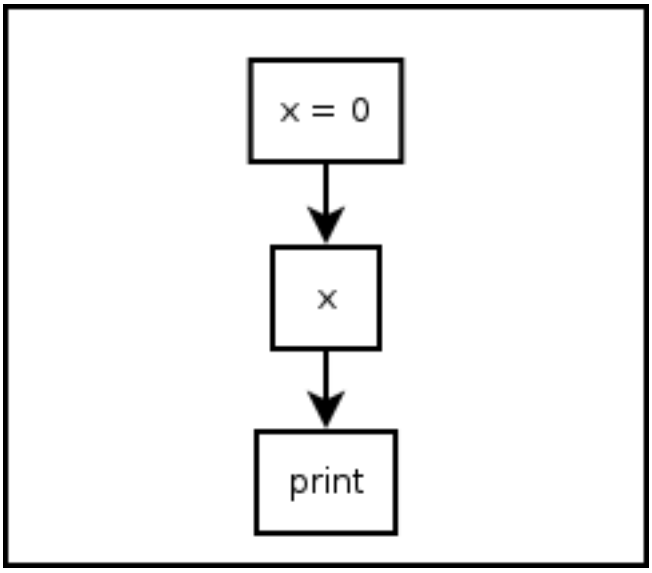
Pythonect supports standard Python types, but with a twist!

Variable `<slug>` in Single Thread

You can define a variable and set its value as follows:

```
[x = 0] -> x -> print
```

Graphically (Visual programming wise), this is represented as:



Both versions print `0`. You can also change its value during the program runtime as follows:

```
[x = 0] -> x -> print -> [x = 1] -> x -> print
```

This will print `0` and `1`, in this order, and in the same thread. Of course you can assign and re-assign any legal Python value, for example:

```
[x = 0] -> x -> print -> [x = "Hello, world"] -> x -> print
```

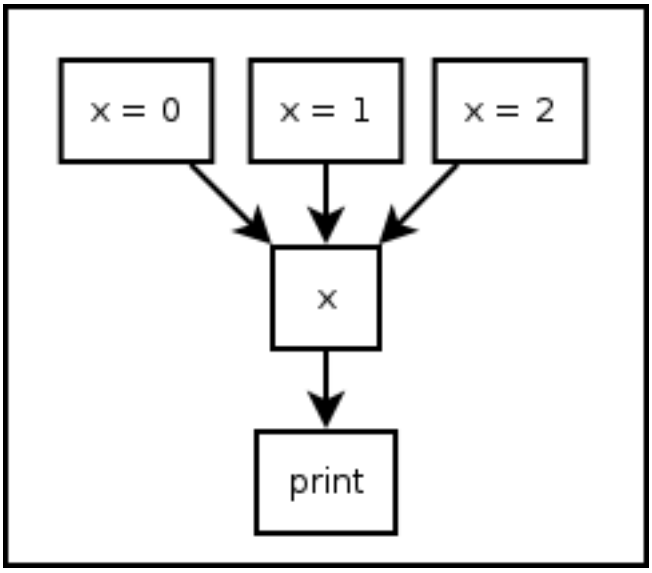
Will print `0`, and `"Hello, world"`

Variable `<slug>` in Multithreading

You can define a variable and set it with multiple values at the same time (each value `map()` 'ed to a different thread) as follows:

```
[x = 0, x = 1, x = 2] -> x -> print
```

Graphically (Visual programming wise), this is represented as:



Both versions will print, each in its own thread, and not necessarily in that order: `0`, `1`, `2`.

Of course you can assign any combinations of Python value and types, for example:

```
[x = 1, x = 1, x = 0.34, x = "Hello, world"] -> x -> print
```

This will print, each in its own thread, and not necessarily in that order: `1`, `1`, `0.34`, and `"Hello world"`.

Predefined Variables

Pythonect predefines two variables: _ and _!

_ as Current Value

The variable underscore (i.e. _) is predefined to be the current value on the flow, for example:

```
1 -> _ + 1 -> print
```

Will print `2`. As _ will be equal `1` after evaluating the `1` expression. Another example is:

```
["To be", "Not to be"] -> print "To be, or not to be? " + _
```

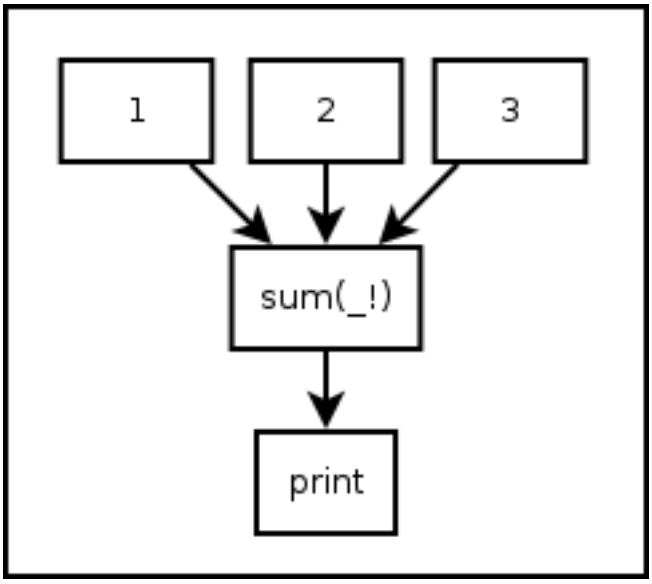
This will print, each in its own thread, and not necessarily in that order: `To be, or not to be?  To be` and `To be, or not to be?  Not to be`

_! as All Current Values

The variable underscore question mark (i.e. `_!`) is predefined to be the list of all the current values in the program (i.e. `reduce()` 'ed), for example:

```
[1,2,3] -> sum(_!) -> print
```

Graphically (Visual programming wise), this is represented as:



Both versions will print `6`. Notice how `sum` became a *reduce*-like function, when it accepted a list of `1`, `2`, `3` and returned `6`. Another example:

```
"Hello, world" -> reversed -> reduce(lambda x,y: x+y, _!) -> print
```

This will print `"dlrow ,olleH"` (i.e. `"Hello, world"` reversed)

1.3.5 Control Flow Tools

Pythonect supports standard control flow tools, but with a twist!

Using Boolean Values as `if` Statement

There's no `if` keyword in Pythonect, instead, boolean values are used to determine whether to terminate or continue the flow.

`True` as Pass-through

Whenever a Python expression or function returns `True` the current value in the flow is pushed to the next operation. For example:

```
1 -> [_ < 2] -> print
```

Will print `1`, because the expression is evaluated to `True` (i.e. 1 < 2). Another example:

```
"Hello, world" -> _.startswith('Hello') -> print
```

Will print `"Hello, world"` because `startswith` method returned `True` (i.e. "Hello, world" string starts with "Hello" string).

`False` as Terminator

Whenever a Python expression or function returns `False` the current flow terminates and returns False. For example:

```
"Hello, world" -> _.startswith('ello') -> print
```

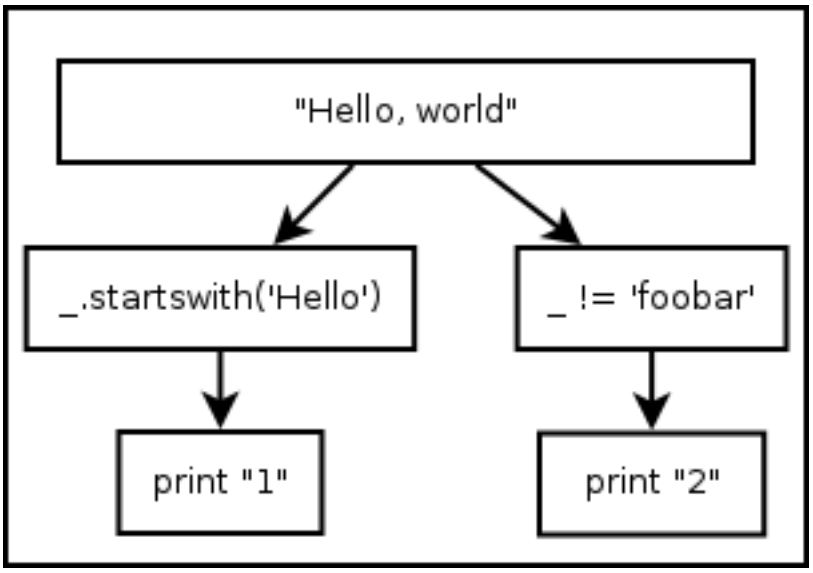
This will not print anything and return `False`.

Using Multithreading as `elif` and `else`

Since there's no `if`, there's also no `elif` or `else`, instead all possible flows are evaulated at once. For example:

```
"Hello, world" -> [[_.startswith('Hello') -> print "1"], [[_ != 'foobar'] -> print "2" ]]
```

Graphically (Visual programming wise), represented as:



Both versions will print (each in its own thread, and not necessarily in that order): `1` and `2`.

Using Iterable Object as `for` Statement

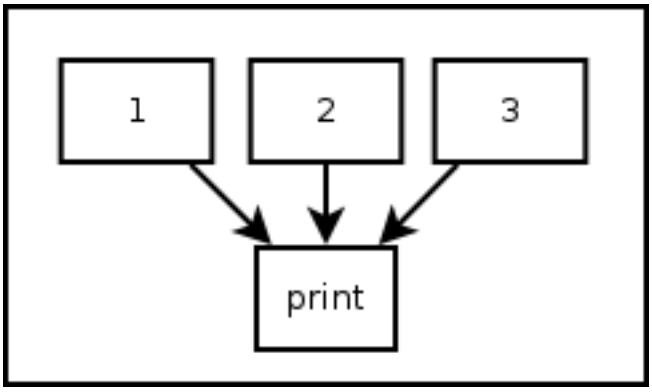
There's no `for` keyword, instead, every Iterable object is treated as a for-loop. For example:

```
[1,2,3] -> print
```

And

```
(1,2,3) -> print
```

Both, Graphically (Visual programming wise) represented as:



Will print `1`, `2`, and `3` (not necessarily in that order). The same applies to generator functions (i.e. `yield`) and Python classes that implement __iter__() method.

The only exception are strings. Pythonect does not iterate Python strings by default. To iterate a string in Pythonect, use Python's built-in iter() function as follows:

```
iter("abc") -> print
```

The code above will print the letters: `a`, `b`, and `c` (not necessarily in that order).

Using Dictionary as `switch` Statement

There's no `switch` keyword in Pythonect, instead, every literal `dict` can be used as a `switch`-like mechanism (without fallthrough). For example:

```
1 -> {1: 'One', 2: 'Two'} -> print
```

Will print `One`. While:

```
3 -> {1: 'One', 2: 'Two'} -> print
```

Will will not print and return `False`

1.3.6 Power Features

This section groups a few of Pythonect power features that one day may get their own section

Autoloader

Pythonect includes an autoloader functionally. You can access any functions, attributes, etc. from a Python module without having to `import` it first. For example:

```
"Hello, world" -> string.split | print
```

Will print (in that order, and in different threads): `Hello,` and `world`. Another example:

```
sys.path -> print
```

Will print (in no particular order) every directory in your `PATH`, as represented in your sys.path.

Using & to Spawn a New Process

Pythonect uses threads by default, but you can switch to processes by using the metacharacter `&`. For example:

```
"Hello, world" -> print &
```

This will print `"Hello, world"` from a new process. You can also mix and match:

```
"Hello, world" -> [print, print &]
```

This will print `"Hello, world"` twice, one from a new thread and the other from a new process.

Notice that switch between thread and process is only valid for the duration of the function call/expression evaluation.

Using @ to Remote Call

Pythonect lets you call functions remotely, by using the metacharacter `@`. To demonstrate this, you will need to use the following simple server (written in Python):

```
from SimpleXMLRPCServer import SimpleXMLRPCServer
from SimpleXMLRPCServer import SimpleXMLRPCRequestHandler


class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)


def say_hello(x):
    return "Hello " + x + " and world"

server = SimpleXMLRPCServer(("localhost", 8000), requestHandler=RequestHandler)
server.register_function(say_hello)
server.serve_forever()
```

Save it as `xmlrpc_srv.py` and run it. This code will run a Simple XML-RPC server that will export a function called `say_hello`.

Now, calling `say_hello` from Pythonect is as easy as:

```
"foobar" -> say_hello@xmlrpc://localhost:8000 -> print
```

This will print `Hello foobar and world`.

The destination hostname can also be the result of a function call, or an expression. For example:

```
"foobar" -> say_hello@"xmlrpc://" + "localhost:8000" -> print
```

Or:

```
"foobar" -> say_hello@"xmlrpc://" + get_free_host() -> print
```

Where `get_free_host()` is a fictional Python function that will return an available hostname from a list of hostnames.

As a loopback, you can use `None` as an hostname to make the call locally. For example:

```
"Hello, world" -> print@None
```

Is equal to:

```
"Hello, world" -> print
```

Both will print `"Hello, world"` locally.

CHAPTER

TWO

API REFERENCE

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 `pythonect` — Parse and execute Pythonect code

This Python module provides the capability to parse and evaluate a string as Pythonect code

`pythonect.`**`parse`**(*source*)

Parse the source into a directed graph (i.e. networkx.DiGraph)

Args: source: A string representing a Pythonect code.

Returns: A directed graph (i.e. networkx.DiGraph) of Pythonect symbols.

Raises: SyntaxError: An error occurred parsing the code.

`pythonect.`**`eval`**(*source*, *globals_={}*, *locals_={}*)

Evaluate Pythonect code in the context of globals and locals.

Args:

source: A string representing a Pythonect code or a networkx.DiGraph() as returned by parse()

globals: A dictionary. locals: Any mapping.

Returns: The return value is the result of the evaluated code.

Raises: SyntaxError: An error occurred parsing the code.

CHAPTER

THREE

ADDITIONAL NOTES

Design notes, legal information and changelog are here for the interested.

3.1 Pythonect Changelog

Here you can see the full list of changes between each Pythonect release.

Ticket numbers in this file can be looked up by visiting http://github.com/ikotler/pythonect/issues/<number>

3.1.1 What's New In Pythonect 0.6?

Release date: 22-Jul-2013

Core and builtins

- Rewrite engine to be both a visual programming language and a text-based scripting language.
- Add support for GNOME Dia (*.DIA) as an Input File Format for Visual Programming
- Add support for Microsoft Visio (*.VDX) as an Input File Format for Visual Programming
- Rewrite Pythonect's Text-based scripting parser (*.P2Y) to use Python's tokenize instead of PLY
- Modify Pythonect's parse() to return a directed graph (i.e. NetworkX.DiGraph)
- Add auto-generated SphinX doc (i.e. doc/)
- Modify NEWS reStructuredText Format to better suit the auto-generated doc

Build

- Modify setup.cfg to stop running tests after the first error or failure

Miscellaneous

- Add examples/ directory with a few example programs

3.1.2 What's New In Pythonect 0.5?

Release date: 24-Apr-2013

Core and builtins

- Issue #71: Unable to run Pythonect in Script Mode
- Issue #72: Can't Load Local Modules with '-m'
- Issue #58: Install argparse if Python 2.6
- Feature #70: Pythonect now supports Max Threads Limit (i.e. '–mt' command-line argument)
- Feature #73: '_' is no longer implicit in the 1st call to eval()

3.1.3 What's New In Pythonect 0.4.2?

Release date: 16-Feb-2013

Core and builtins

- Feature #61: Interpreter supports command line '-c', '-i' and '-m'
- Enhancement #68: Improved Interpreter Banner
- Enhancement #67: args **globals_** and **locals_** of eval() are now optional
- Feature #66: Within Pythonect Program: eval() now takes Pythonect code and __eval__() takes Python
- Refactor __run() [Guy Adini]
- Feature #65: Pythonect now supports PyPy
- Feature #55: Pythonect now supports Python 2.6
- Issue #48: 'print "B" in "ABC"' and 'print 2 is 2' throws Exception
- Issue #60: "copyright", "license", and "credits" are not of Pythonect
- Issue #62: Parameterless functions are now handled properly
- Issue #63: "quit" and "exit" raises ValueError: I/O operation on closed file
- Issue #64: Interpreter command line option '–version'/'-V' output wrong banner
- Issue #69: print/**print_** can not be overridden by locals or globals value

Build

- Switched to nosetests (+ coverage)
- Issue #49: zip_safe is not False by default

3.1.4 What's New In Pythonect 0.4.1?

Release date: 03-Sep-2012

Core and builtins

- PEP8 Fixes
- PEP 3110 Fixes
- Added Travis CI Support
- Issue #38: No docstrings for eval(), parse(), and Pythonect module
- Issue #39: eval_test_gen.py fails due to incorrect import
- Issue #41: Pythonect split() renamed to parse() to better fit it's purpose
- Issue #42: Pythonect fails on Python implementations that do not include the multiprocessing module
- Enhancement #45: Dict can now be used as a return value, only literal dict will be treated as switch
- Issue #47: Pythonect parse() is not included in the testsuite

Build

- Issue #43: Pythonect unittest runner is not cross-platform
- Issue #44: Warnings during installation due to MANIFEST.in

3.1.5 What's New In Pythonect 0.4?

Release date: 09-Aug-2012

Core and builtins

- Issue #31: Synchronous/Asynchronous is not enforced when execution return value is callable and iterable
- Issue #32: Script can't accept command line args
- Issue #34: Script file can't contain Backslash
- Feature #34: Interpreter (in Interactive mode) now logs commands for further use
- Feature #35: Pythonect module now exports split() function to parse Pythonect code
- Feature #36: Backticks can be used to evaluate a Pythonect expression

Miscellaneous

- Removed eXecute bit from pythonect/__init__.py and pythonect/internal/__init__.py
- Reorganized Pythonect module structure (pythonect.eval.eval is now pythonect.eval)

3.1.6 What's New In Pythonect 0.3.1?

Release date: 14-Jul-2012

Core and builtins

- Issue #25: Pythonect package namespsace (when importing from Python) is polluted
- Issue #26: Odd Single quote char breaks double quote String (and vice versa)
- Issue #27: Multiprocessing is not working with multi-threading
- Issue #28: Autoload always throws NameError regardless to the actual Exception type
- Issue #29: Preprocessor breaks on a List with Function Call that contains String
- Issue #30: Preprocessor incorrectly process non-String literals in List

3.1.7 What's New in Pythonect 0.3?

Release date: 20-Jun-2012

Core and builtins

- Feature #13: Improved print function
- Feature #15: Implemented Stateful Interpreter
- Feature #17: Remote procedure URL can be an expression
- Feature #18: Implemented Multiprocessing
- Feature #20: Backslash can be used to join two or more physical lines into a logical line
- Feature #22: Implemented None as pseudo remote protocol / URL
- Issue #14: Print does not act as a pass-through statement
- Issue #16: TypeError Exceptions are not been displayed
- Issue #19: Autloading is not working in a statement
- Issue #21: Preprocessor breaks on a List with a String that contains comma

Build

- Issue #12: No newline at the end of _version.py (PEP8)

3.1.8 What's New in Pythonect 0.2.1?

Release date: 27-May-2012

Core and builtins

- Issue #9: Autoload won't load modules from current working directory
- Issue #11: Autoload parses name incorrectly if in a list or tuple

3.1.9 What's New in Pythonect 0.2?

Release date: 30-Apr-2012

Core and builtins

- Feature #8: Implemented Autoloading.
- Feature #7: Python built-in dictionary can be used as a switch statement.
- Issue #6: Interpreter prints Strings without quotes
- Issue #5: Interpreter lags when pressing Enter key multiple times

Build

- Issue #4: Pythonect reports incorrect version if installed via pip/sdist.

3.1.10 What's New in Pythonect 0.1.1?

Release date: 18-Apr-2012

Core and builtins

- Issue #3: Check that the program return value is not None before printing it

Build

- Issue #1: Removed import from __init__ to avoid PLY imports via setup.py.

Miscellaneous

- Add NEWS file

3.1.11 What's New in Pythonect 0.1?

Release date: 01-Apr-2012

Everything :-)

3.2 License

Pythonect licensed under BSD 3-Clause:

```
Copyright (c) 2012-2013, Itzik Kotler
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
```

```
      documentation and/or other materials provided with the distribution.

    * Neither the name of the author nor the names of its contributors may
      be used to endorse or promote products derived from this software without
      specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```