Wi-Fi Settings

SSID: AcroboticGuest

Password: 13meetup37

A Gentle Introduction to Programming Using Python

ACROBOTIC industries

October 20, 2015

Francisco Zabala

Downloading this presentation

This repository Search

Pull requests Issues Gist

acrobotic / Ai_Intro_RPi

Navigate to:

https://github.com/acrobotic/Ai_Intro_Python

Unwatch 3 Star 1 Fork 0

Code, wiring diagrams, and presentation slides for introductory activities on the Raspberry Pi single-board computer. — Edit

8 commits 1 branch 0 releases 1 contributor

Branch: master Ai_Intro_RPi / +

.

themakerbro authored 14 days ago latest commit 770a4efb02

activity_01	done	14 days ago
activity_02	done	14 days ago
activity_03	done	14 days ago
activity_04	adding activities 3 and 4	14 days ago
activity_05	activities completed	14 days ago
activity_06	activities completed	14 days ago
activity_07	activities completed	14 days ago
activity_08	done	14 days ago
presentation	.	14 days ago

Code

Issues 0

Pull requests 0

Wiki

Pulse

Graphs

Settings

HTTPS clone URL

https://github.com/

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

Who is this class for?

People looking to learn programming (new or re-visiting after some time)

People looking for professional development (change or enhance careers)

Students

IT professionals

Class outline

Duration: 3 hrs; Difficulty: Low

Motivation

Why learn to code?

Why Python?

Who uses Python?

Getting Started

Raspberry Pi, Linux, Python interpreter, and Integrated Development Environment (IDLE)

Using Python Interactively

Python as a calculator

Basic data types, assignment, and variables

Interactive programs: input and output

Control of flow (choice and loops)

Compound data types (lists and dictionaries)

Functions

Built-in modules

Writing and Running Python Scripts

Working with Python scripts

Custom modules

Reading and writing files

Applications

Plotting data with *matplotlib*

Running a simple web application with Flask

Motivation

Why learn to code?

Programming is a tool that allows you to tackle and (hopefully) solve many kinds of problems.

We’re surrounded by data (digital age), programming allows to interact with data in meaningful and efficient ways.

Automating repetitive tasks in your computer (e.g., search and replace multiple docs, renaming photos).

Python vs. Shell Scripts vs. C/C++/…

Why Python?

Python is a *general purpose* programming language created by Guido Van Rossum.

One of the *easiest* languages to learn/teach (friendly syntax).

Wealth of *free* tools to start developing in Python.

Large community *support* (Q&A, extensive collection of libraries).

Concepts applicable to other programming languages like C#, Perl, etc.

Ranked among the *top eight* most popular programming languages in the world.

Why Python?

Language Features

Python is an interpreted language (no need to compile).

Python is dynamically typed (no need to declare data types).

Statement grouping is done by indentation instead of beginning and ending brackets (readability!)

Paired with a full-featured scripting interpreter.

Typical Applications

Used to create many things such as web and desktop applications, games, analyzing and visualizing data, and much more.

Why Python?

Typical Applications

Used to create many things such as web and desktop applications, games, analyzing and visualizing data, and much more.

Who uses Python?

NASA:

Google:

Pinterest:

Instagram:

Kickstarter:

The Onion:

Getting Started

Raspberry Pi Fundamentals

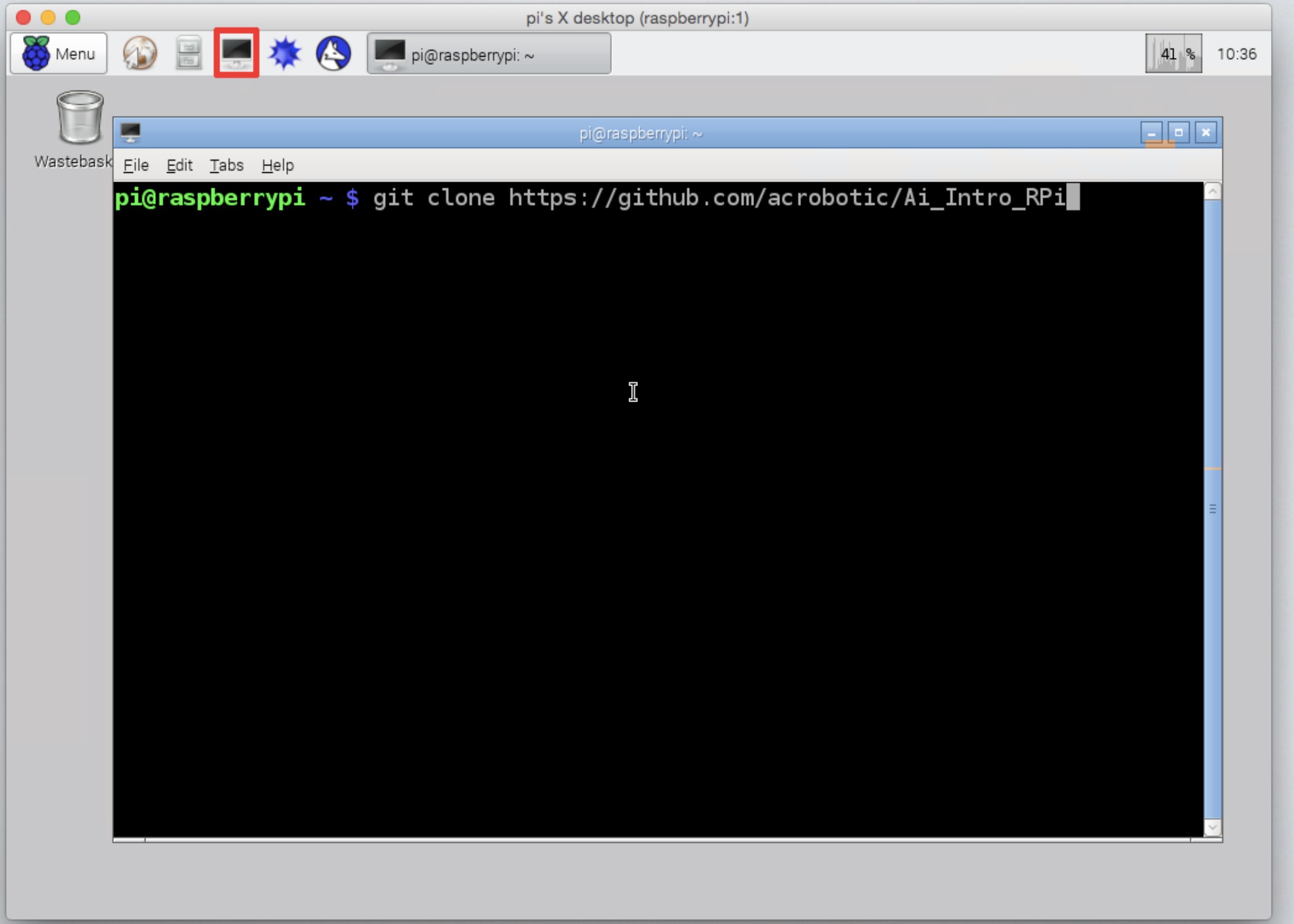
Getting Started with Raspberry Pi:

http://learn.acrobotic.com

https://www.youtube.com/watch?v=ZJU7mns3juc

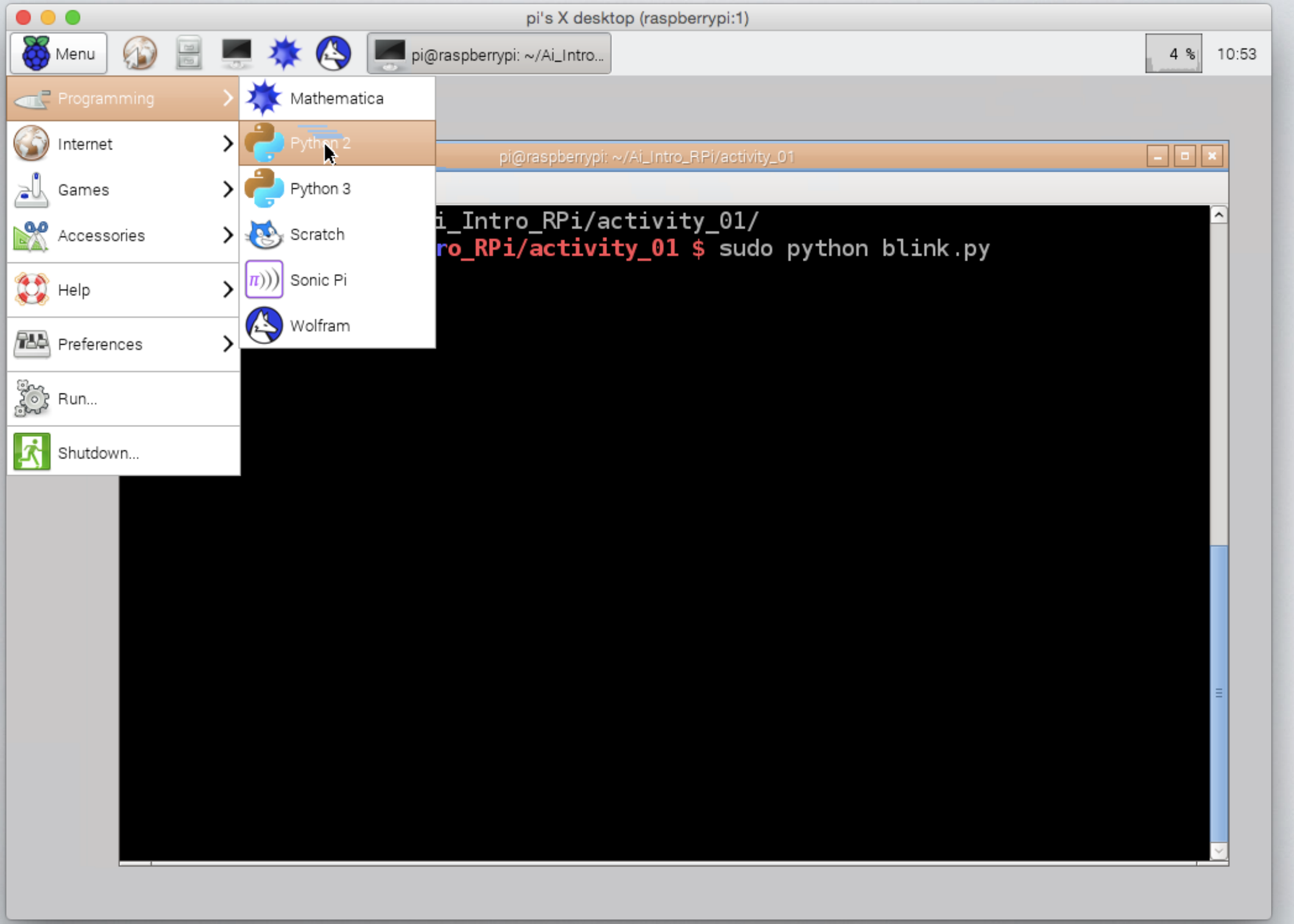
Linux Fundamentals

Accessing the Terminal and getting the activities code



Linux Fundamentals

Opening scripts in Python's "Integrated DeveLopment Environment"



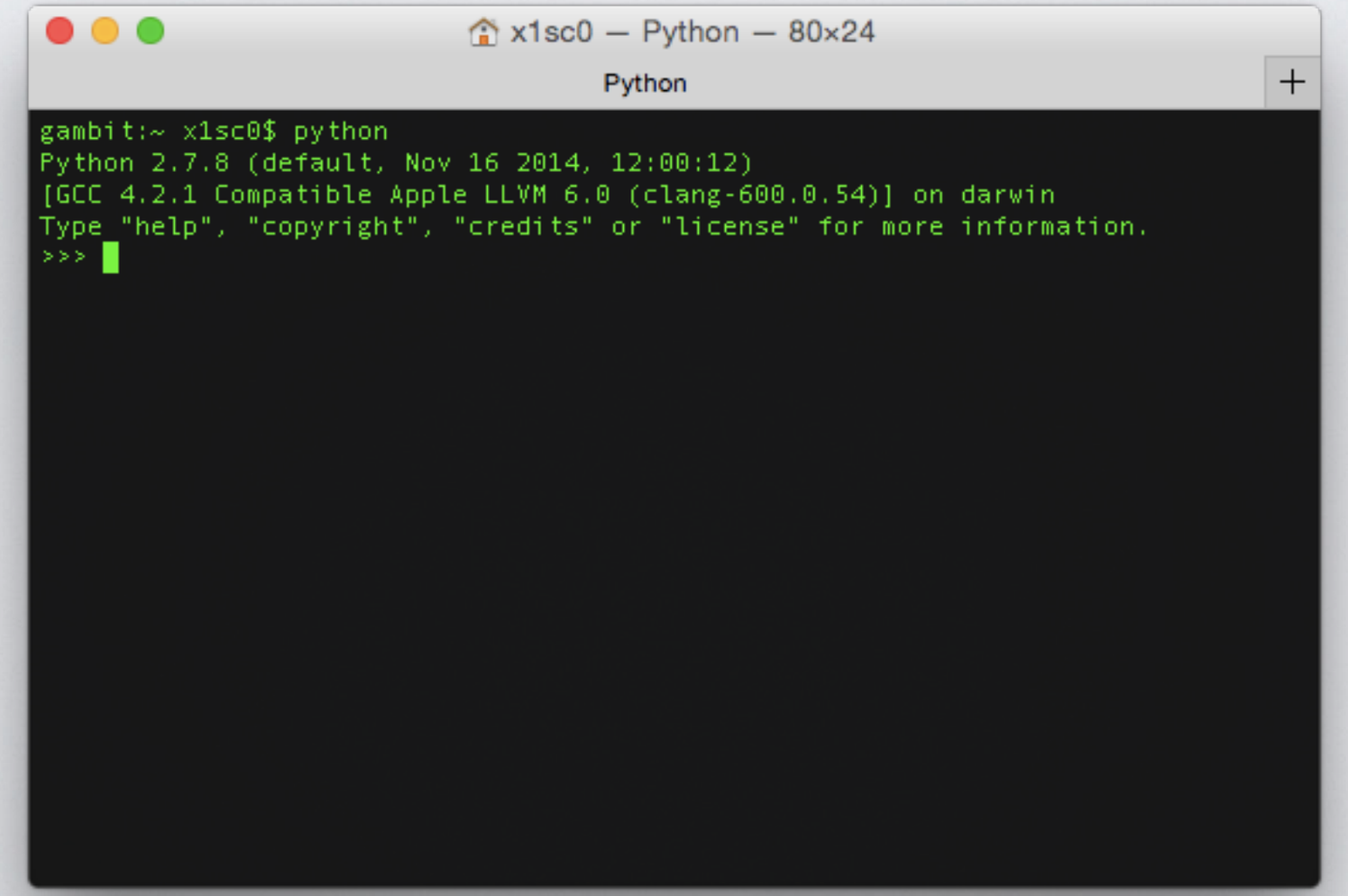
Using Python Interactively

The Python Interpreter

The interpreter is the program that allows you to run ‘unpacked’ Python code on your computer.

It can be run in interactive (calculator) mode by issuing the command:

```
python
```

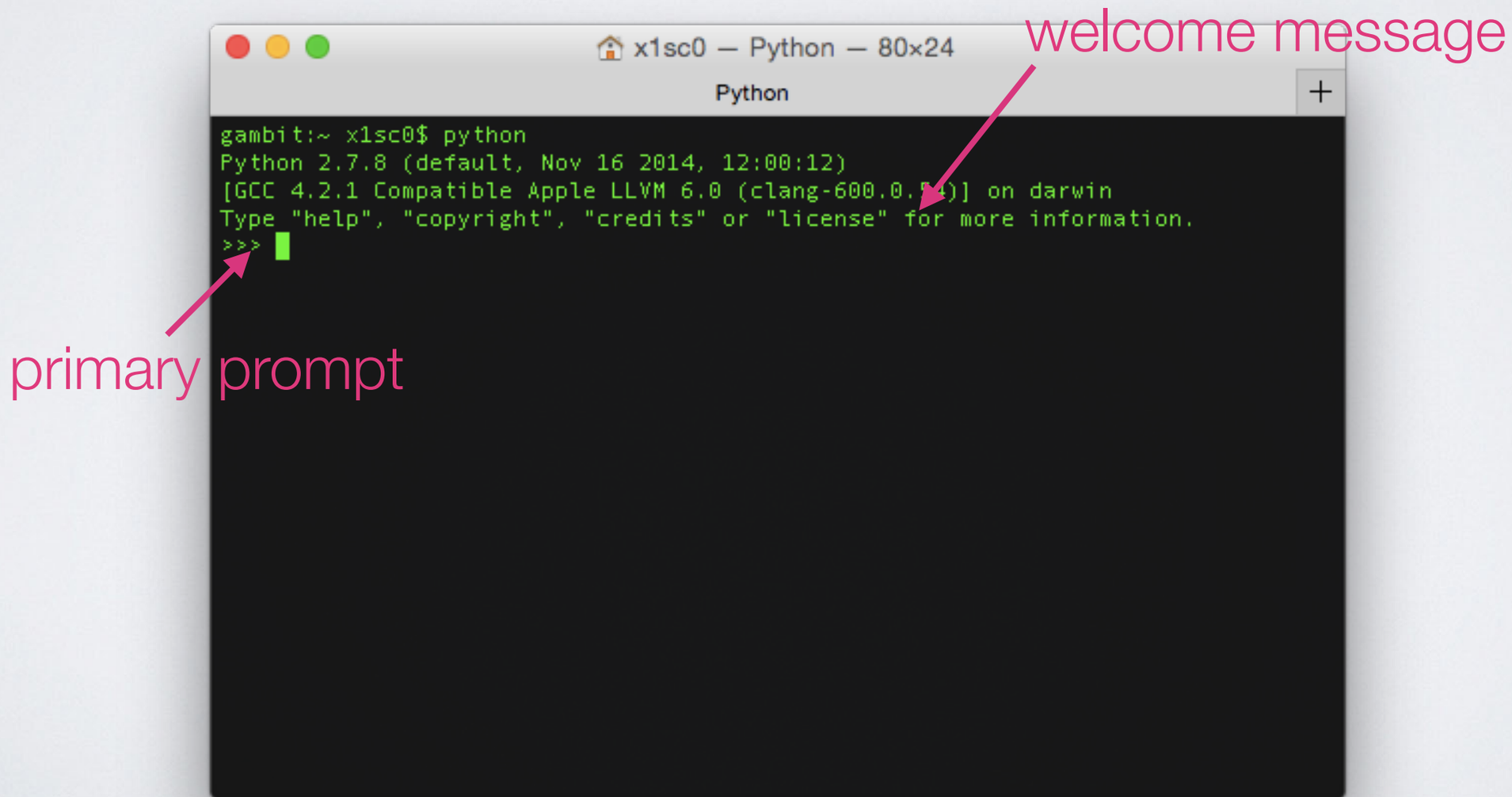


The Python Interpreter

In interactive mode the interpreter prompts for the next command with the primary prompt (>>>).

For continuation lines the interpreter prompts with the secondary prompt (...).

The interpreter prints a welcome message (version number and copyright)



Numbers

In interactive mode Python behaves as a calculator.

It ignores whitespace except for indentation

We need to be careful with operations between different **data types**.

x1sc0 — Python — 80×24

Python

```
gambit:~ x1sc0$ python
Python 2.7.8 (default, Nov 16 2014, 12:00:12)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.54)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 2
4
>>> 50-1
49
>>> 40*11
440
>>> 8 / 5.0
1.6
>>> 8 / 5
1
>>>
```

Data Types

Basic

Integers: **`1`**, **`2`**, ...

Floats: **`1.25`**, **`3.14159`**, ...

Booleans: **`True`**, **`False`**

Compound

Lists: **`[1,2,3]`**

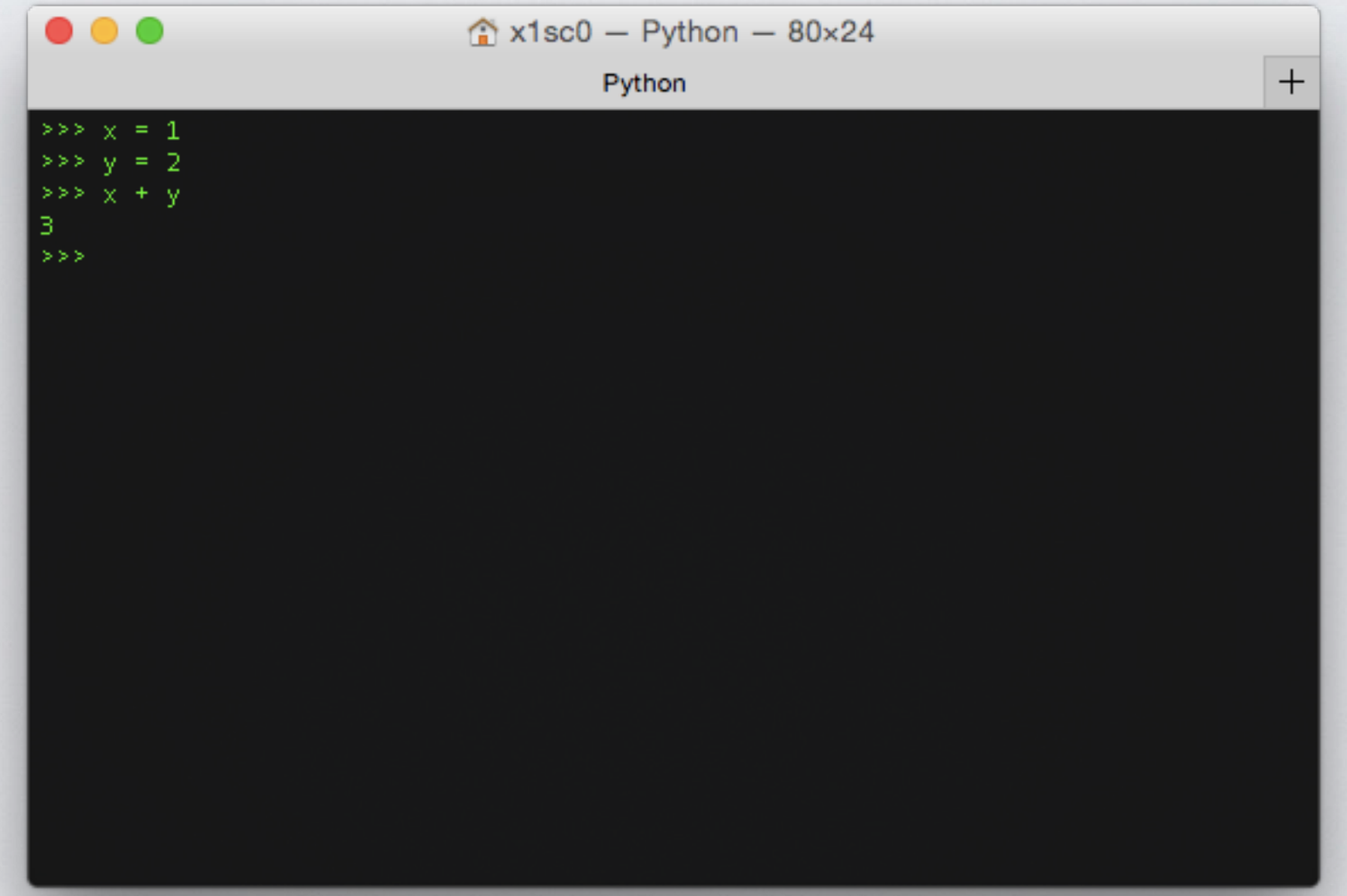
Dictionaries:

Assignment and Variables

Variables can store values and we tend to use them similarly than in math!

The equal sign (=) is used to assign a value to a variable.

Variables are quite powerful, they can store any data type!

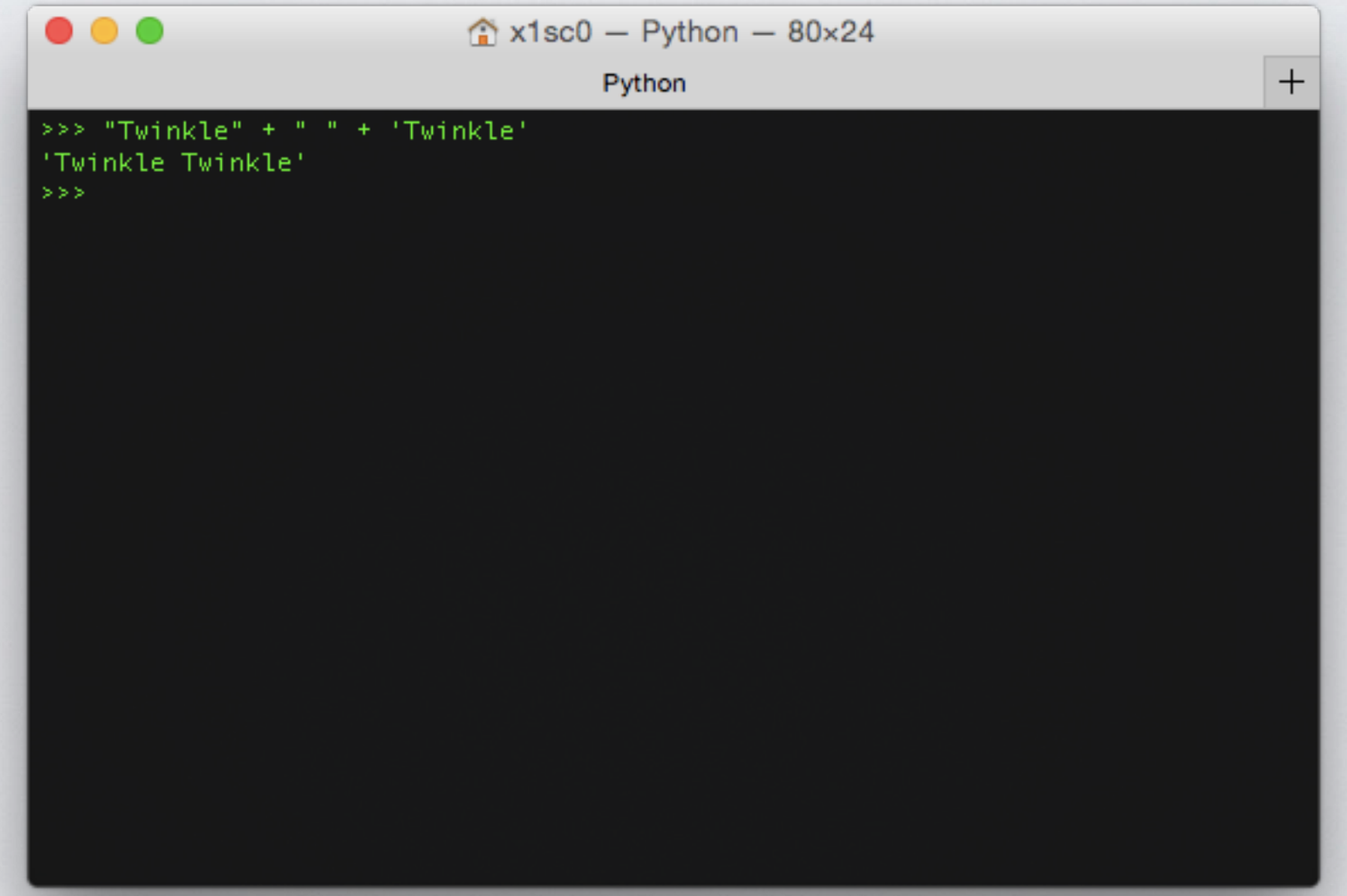


Note: afterwards, no result is displayed before the next interactive prompt.

Working with Strings

In computer programming, a string is traditionally a sequence of characters.

In Python, strings are enclosed in either single (`'...'`) or double quotes (`"..."`) with the same result.



Working with Strings

The **+** operator concatenates strings

The ***** operator repeats strings

Numbers and Strings cannot be concatenated (different data types)

x1sc0 — Python — 80×24

Python

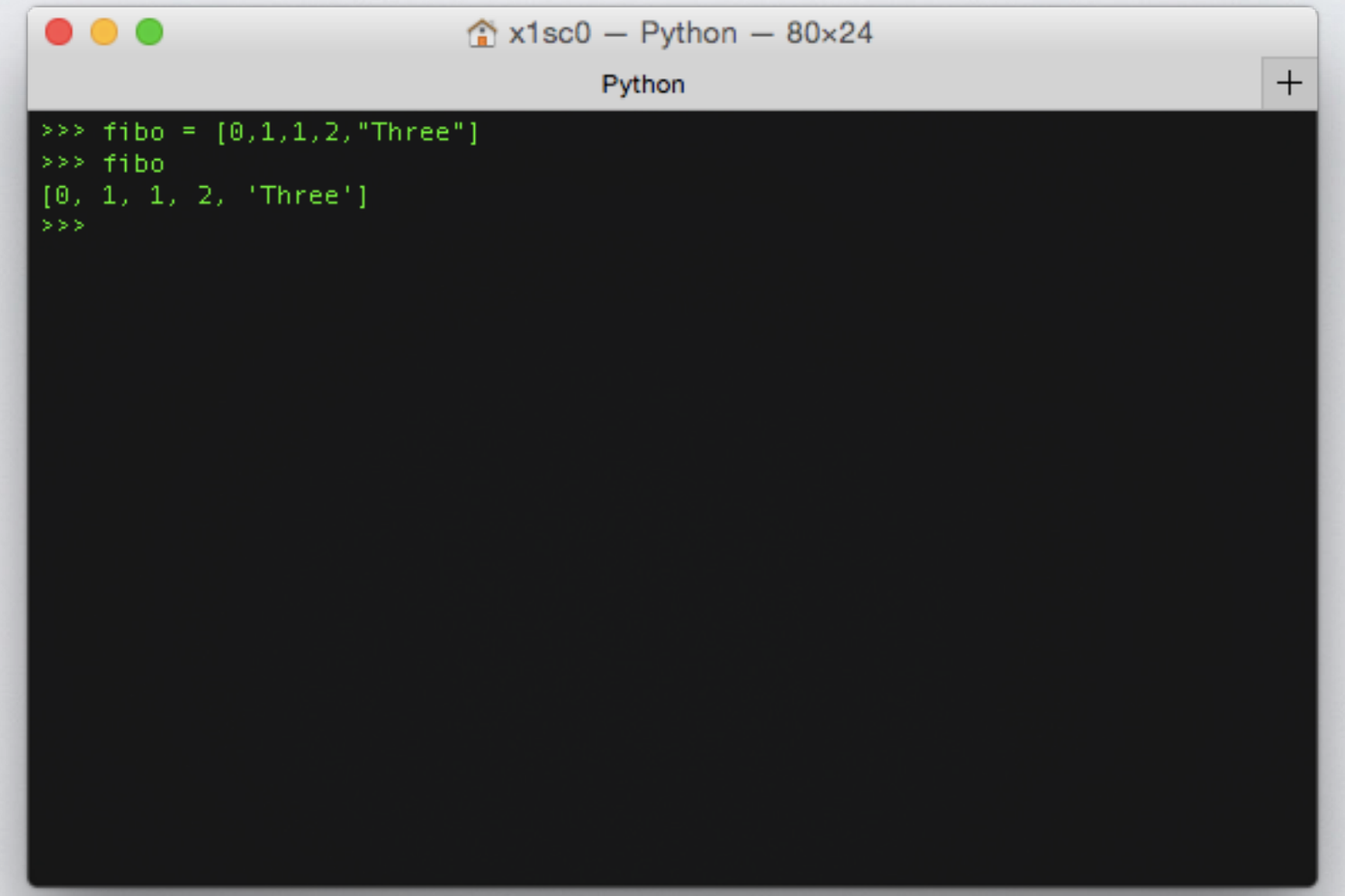
```
>>> "Twinkle" + " " + 'Twinkle'
'Twinkle Twinkle'
>>> a = "Little "
>>> b = 'Star'
>>> a+b
'Little Star'
>>> a+1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> a + "1"
'Little 1'
>>>
```

Lists

Python knows a number of compound data types, used to group together other values. Lists are the most versatile.

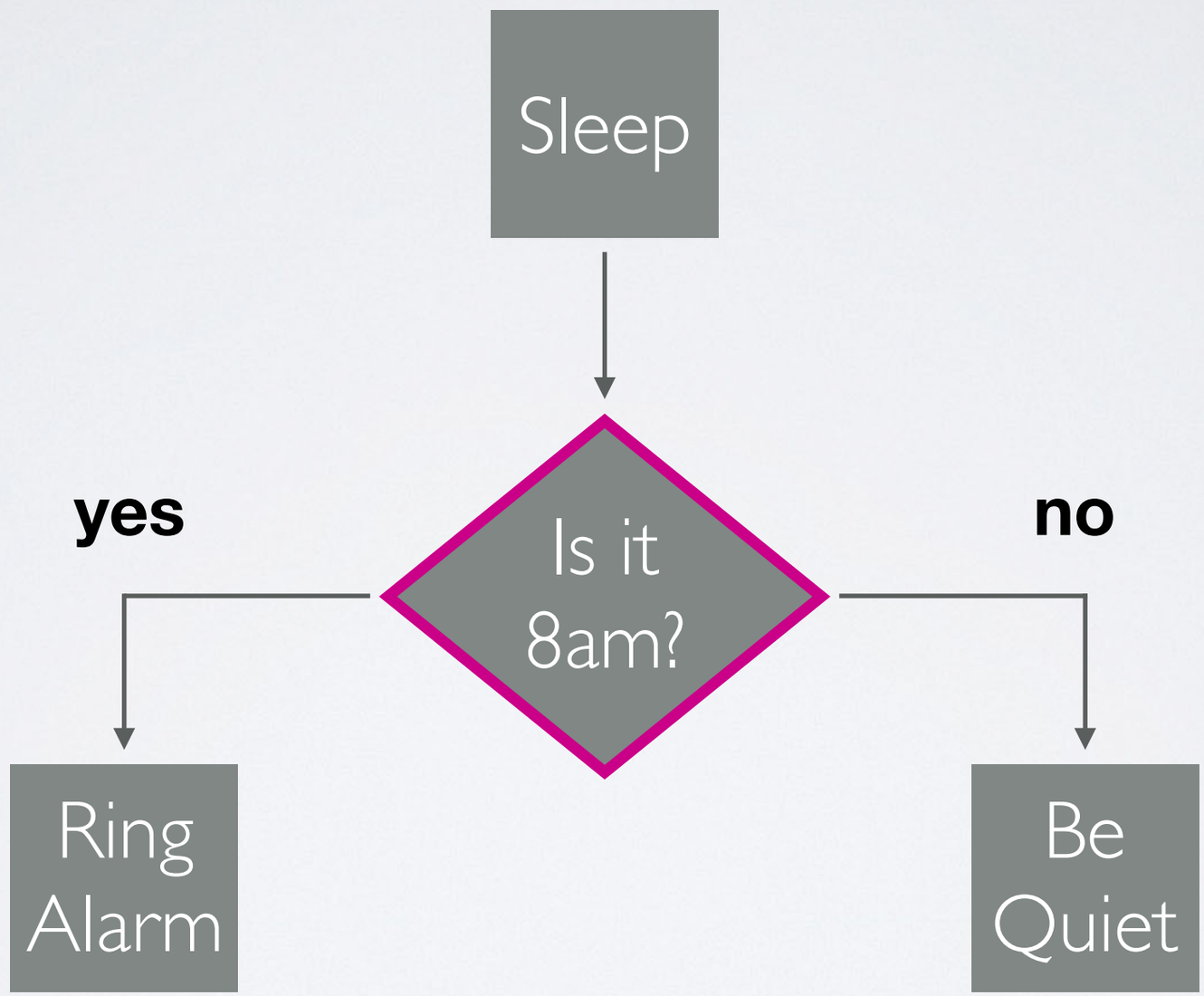
Lists can be written as a comma-separated list of values in square brackets.

Lists might contain items of different types (usually they’re of the same type).



Truth Value Testing

Truth value testing is typically used as a condition to control the flow of the program.



Example: basic alarm clock

Boolean Operations

Boolean operations are a form of algebra in which all values are reduced to either TRUE or FALSE.

Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then `True`, else `False`

Python considers any value to be TRUE (from a Boolean perspective) except:

`False`

`None`

Any empty sequence, for example, **`''`**, **`()`**, **`[]`**.

Zero of any numeric type, for example, **`0`**, **`0L`**, **`0.0`**, **`0j`**.

Comparisons

Comparison (or relational) operators *compare* the values on either sides of them to decide determine their relation.

When comparisons are evaluated they return truth values (i.e., **`True`** or **`False`**)

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Control of Flow: Loops

Ordinarily, code is executed from the first line going downward. Control of Flow refers the use of code that changes the order in which statements are executed.

The `while` statement

The **`while`** statement is used for repeated execution as long as an expression is logically true.

The `for` statement

The **`for`** statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

Control of Flow: Choice

The `if` statement

The **`if`** statement is used for performing different computations or actions depending on whether a condition evaluates to true or false

The general form of the **`if`** statement in Python looks like this:

```
if condition_1:
    statement_block_1
elif condition_2:
    statement_block_2
else:
    statement_block_3
```

Functions

Functions are useful bits of work encapsulated and given a name.

Typically functions operate on something (arguments).

`help(dir)`

```
x1sc0 — less — 80×24
less
Help on built-in function dir in module __builtin__:

dir(...)
    dir([object]) -> list of strings

    If called without an argument, return the names in the current scope.
    Else, return an alphabetized list of names comprising (some of) the attribut
es
    of the given object, and of attributes reachable from it.
    If the object supplies a method named __dir__, it will be used; otherwise
    the default dir() logic is used and returns:
      for a module object: the module's attributes.
      for a class object:  its attributes, and recursively the attributes
        of its bases.
      for any other object: its attributes, its class's attributes, and
        recursively the attributes of its class's base classes.
(END)
```

Functions

The Python interpreter has a number of *functions* built into it that are always available.

		Built-in Functions		
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

Constants

The Python interpreter has a number of *constants* built into it that are always available.

`False, True, None,` …

The Python interpreter has a number of *constants* built into it that are always available.

`dir(__builtin__)`

The built-in function **`dir()`** if called without an argument, return the names in the current scope. The argument **`__builtin__`** allows it to return the names and functions built into the interpreter.

Modules (Built-in)

Python comes with a library of standard modules (see Python Library Reference). Some are built-in (written in C) whereas others are written in Python.

Some modules are built into the interpreter (e.g., sys) that allow us to perform system operations.

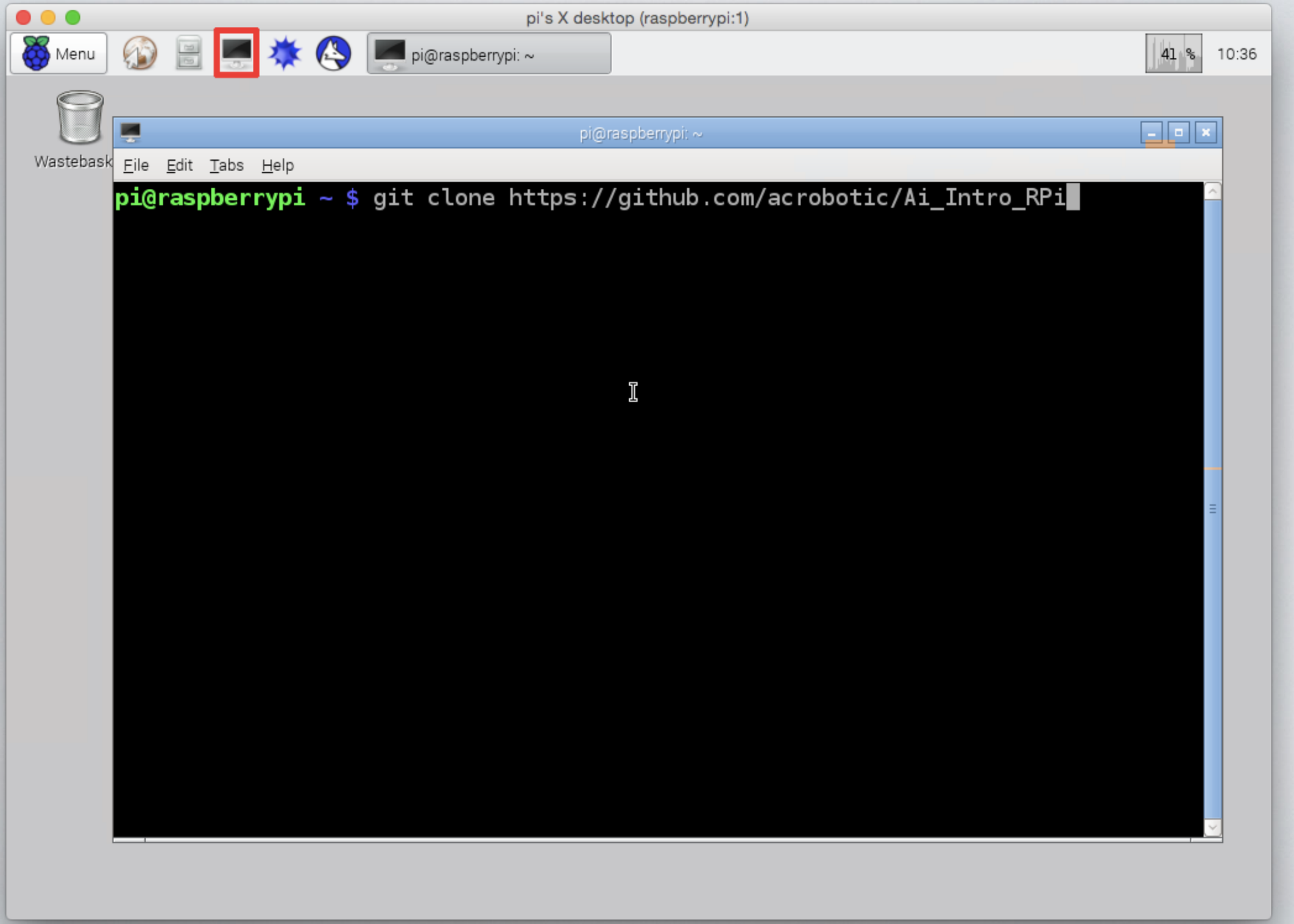
When imported, modules add functionality not readily available in the core Python implementation.

```
import time
time.sleep(5);\
print "hello world"
```

Writing and Running Python Scripts

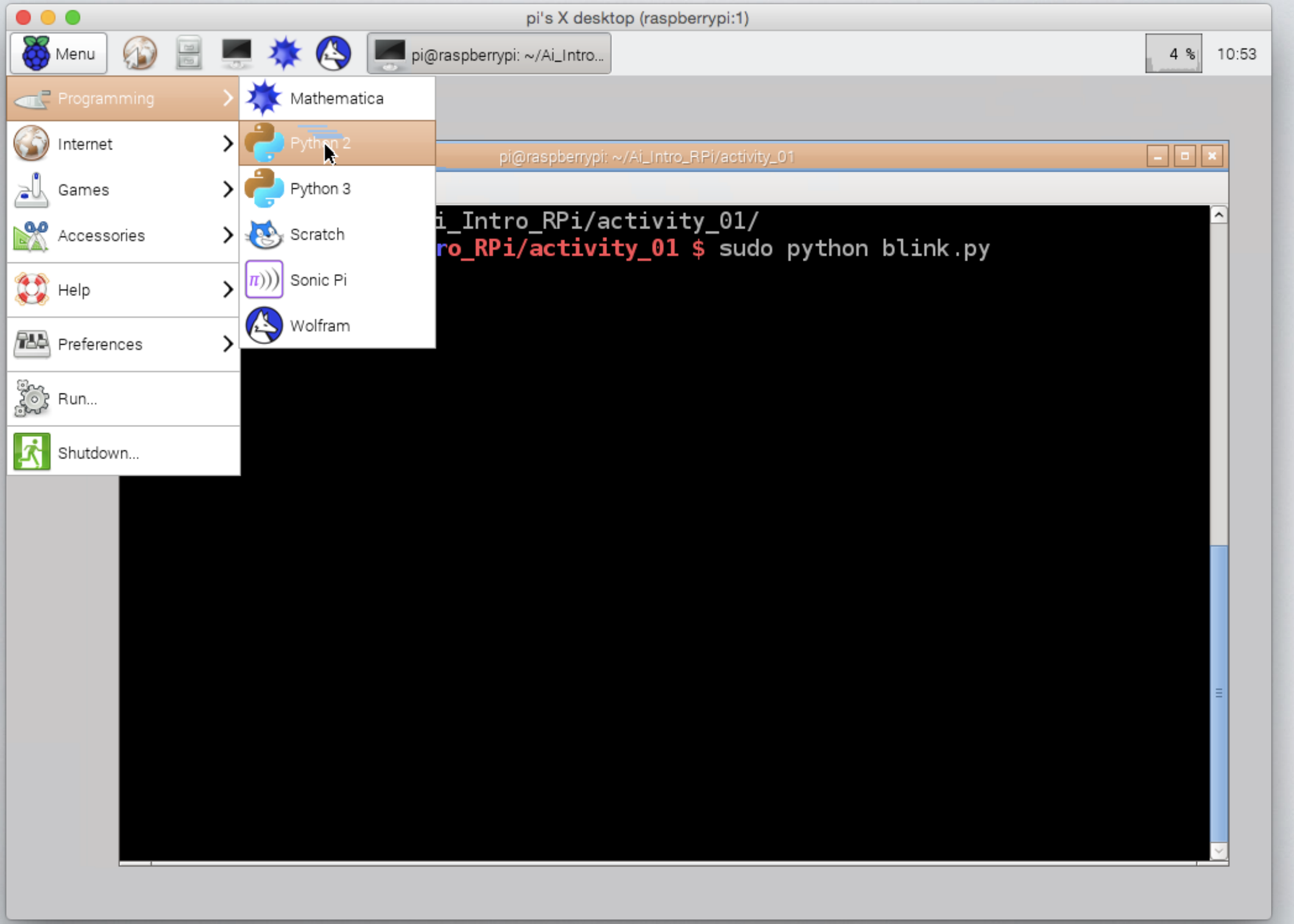
Linux Fundamentals

Accessing the Terminal and getting the activities code



Linux Fundamentals

Opening scripts in Python's "Integrated DeveLopment Environment"



Scripts

Scripts and modules are one and the same from Python's point of view. Both 'scripts' and 'modules' are executable and importable.

We call 'scripts' pieces of code that are directly executable (run by itself).

We call 'modules' pieces of code that are *imported* by other pieces of code.

Modules typically won't do anything (or will just run their unit tests) when executed directly.

Importing code designed to be a script will cause it to execute.

We use the conditional **`if __name__ == "__main__":`** the current file is being executed.

Scripts

Getting some hands-on practice:

codecademy

Intro to Python

Intro to Python

Cube

Exercise goal: Practice printing and manipulating numbers.

A Python interpreter is a convenient calculator!

So far, we have a variable called `favorite_number`.

Instructions

Your job:

use Python to `print` the result of calculating `favorite_number` to the third power.

Add your `print` statement below the existing code, on line 2.

Q&A Forum | Glossary

script.py

```
1  favorite_number = 111
```

http://j.mp/ai-intro-python

Save & Submit Code | Reset Code

1. Cube

Modules (Custom)

A module is a file containing Python definitions and statements.

The file name is the module name with the suffix **.py** appended. (e.g., filename.py)

Within a module, the module's name (as a string) is available as the value of the global variable **`__name__`**.

Modules (Custom)

Let's write our first custom module! Create the following files in your system:

hello.py

```
import time
def hello_function(sleep_time):
    time.sleep(sleep_time)
    print "hello world"
```

main.py

```
from hello import hello_function

hello_function(2)
```

Applications

The Python Package Index (PyPI)

The Python Package Index is a repository of software for the Python programming language.

https://pypi.python.org/pypi

python™

search

» Package Index

PACKAGE INDEX »

- Browse packages
- Package submission
- List trove classifiers
- List packages
- RSS (latest 40 updates)
- RSS (newest 40 packages)
- Python 3 Packages
- PyPI Tutorial
- PyPI Security
- PyPI Support
- PyPI Bug Reports
- PyPI Discussion
- PyPI Developer Info

ABOUT »

NEWS »

DOCUMENTATION »

DOWNLOAD »

COMMUNITY »

FOUNDATION »

CORE DEVELOPMENT »

PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **67937** packages here.

To contact the PyPI admins, please use the Support or Bug reports links.

Not Logged In

Login
Register
Lost Login?
Use OpenID

Status

Nothing to report

Get Packages

To use a package from this index either "pip install *package*" (get pip) or download, unpack and "python setup.py install" it.

Package Authors

Submit packages with "python setup.py upload". The index hosts package docs. You may also use the web form. You must register. Testing? Use testpypi.

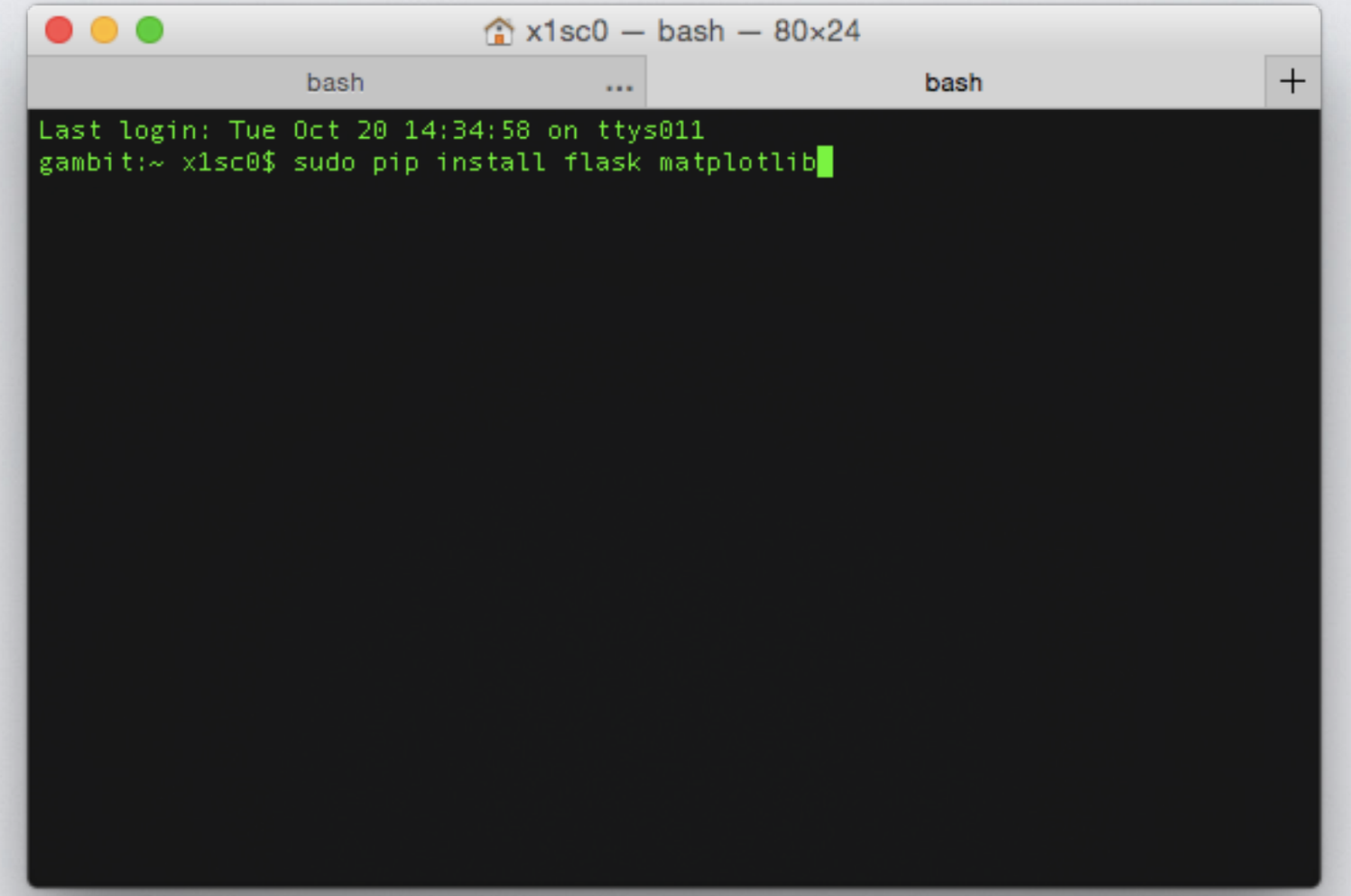
Infrastructure

To interoperate with the index use the JSON, OAuth, XML-RPC or HTTP interfaces. Use local mirroring or caching to make installation more robust.

Updated	Package	Description
2015-10-21	toil 3.1.0a1.dev48	Pipeline management software for clusters.
2015-10-21	django-knob 1.1	A Django reusable application that performs remote configurations on multiple devices, distributing the operations using Celery.
2015-10-20	song2 0.1.0	Typesafe/Immutable schema for dict object
2015-10-20	django-influxdb-metrics 1.2.1	A reusable Django app that sends metrics about your project to InfluxDB
2015-10-20	luigi-monitor 0.2.2	Send summary messages of your Luigi jobs to Slack.
2015-10-20	flask-autorouter 0.1.1	a utility for generating flask URL routing
2015-10-20	django-templatetags 1.1	Custom template tags for notification
2015-10-20	djangorecipe 2.1.2	Buildout recipe for Django
2015-10-20	SciSalt 1.6.1	Tools to make scientific data analysis easier

The Python Package Index (PyPI)

We use either **pip** or **easy_install** (package managers) to install any of the 3rd-party modules available on PyPI

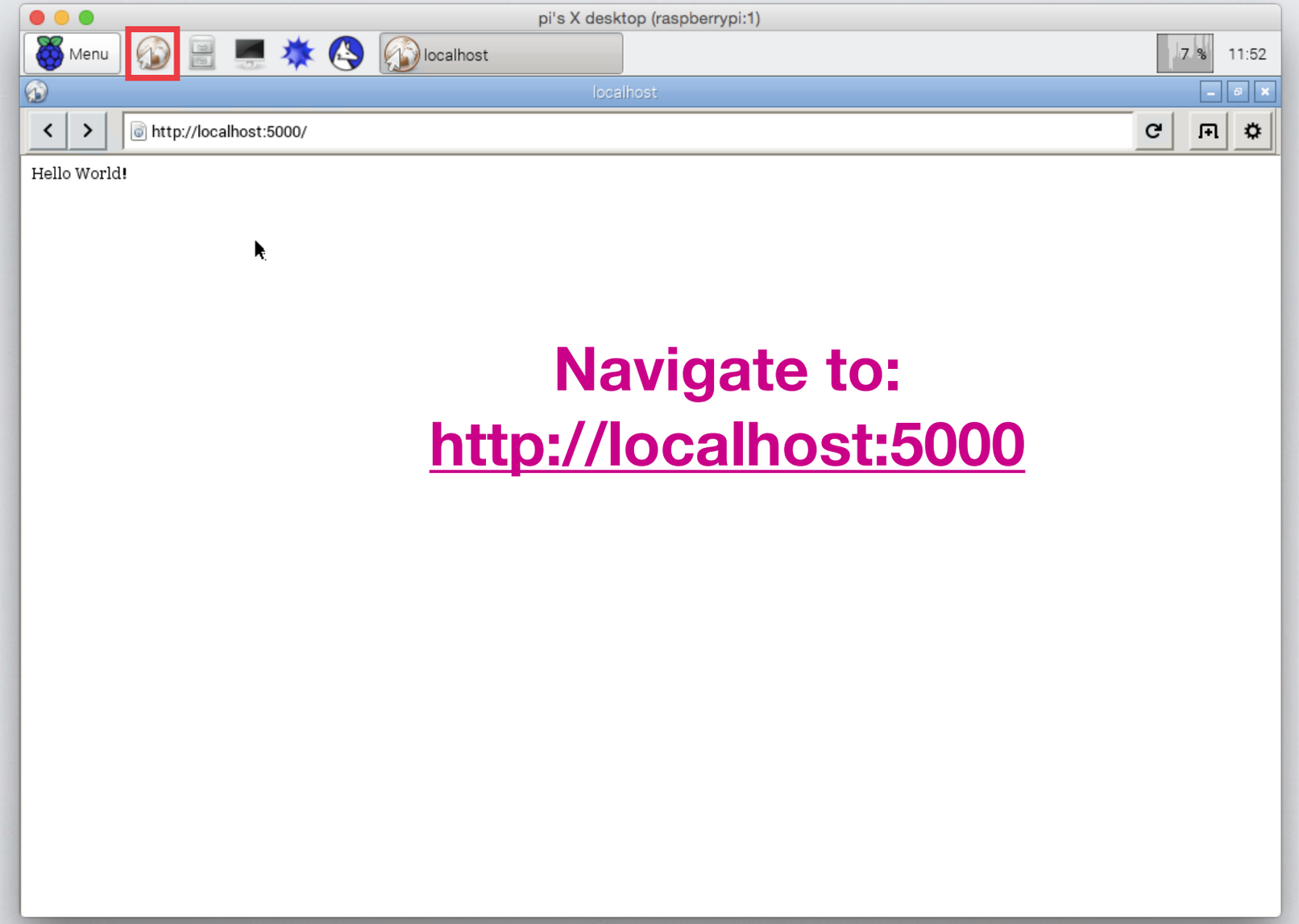


Web Application With Flask

```
cd ~/Ai_Intro_Python/webapp

sudo pip install flask

sudo python webapp.py
```



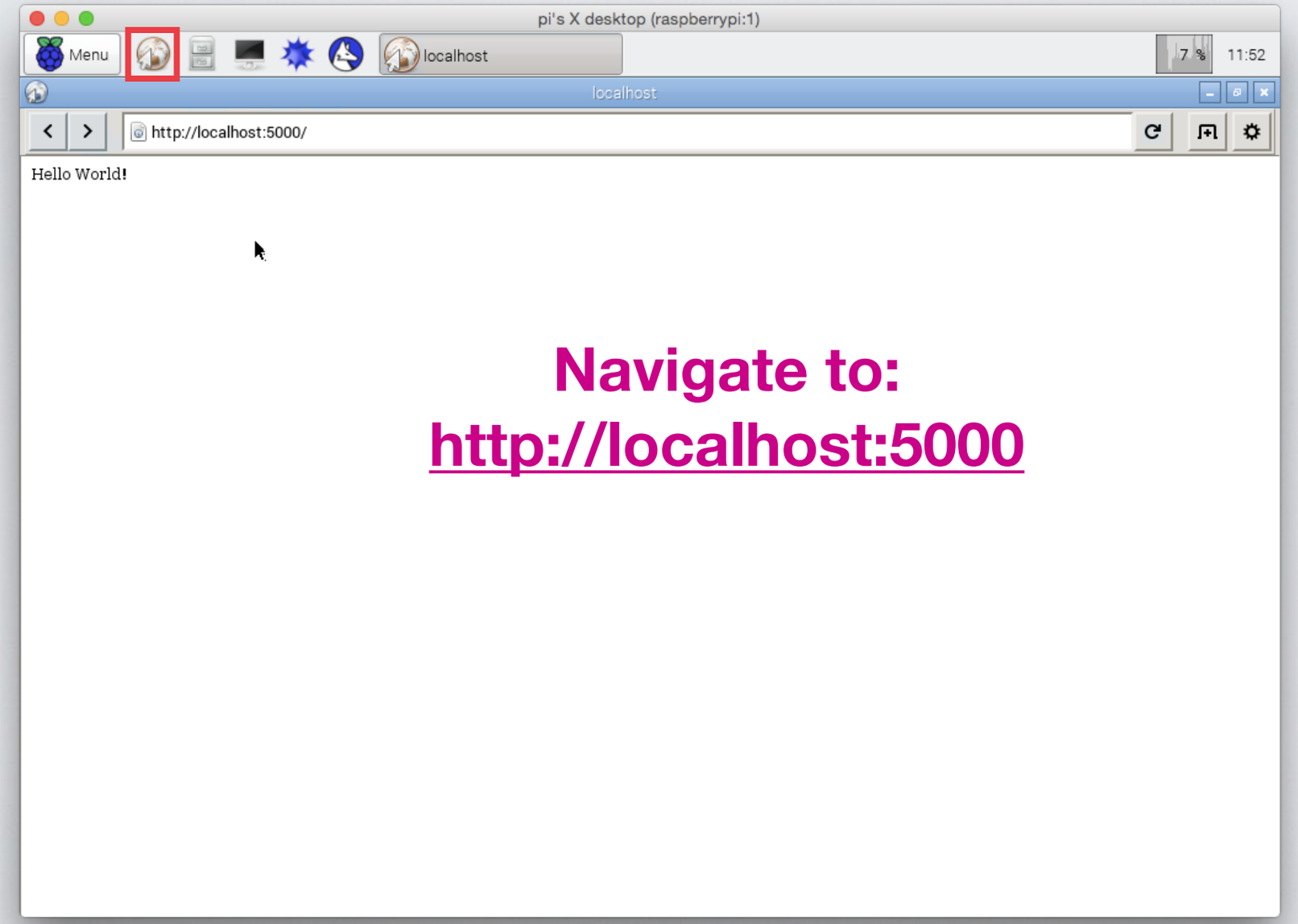
Navigate to:
http://localhost:5000

Web Application With Flask

```
cd ~/Ai_Intro_Python/dataviz

sudo pip install matplotlib numpy

python dataviz.py
```



Navigate to:
http://localhost:5000