



jupyter

jupyter

jupyter

tutorialspoint

SIMPLY EASY LEARNING

jupyter

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

Project Jupyter is a comprehensive software suite for interactive computing, that includes various packages such as Jupyter Notebook, QtConsole, nbviewer, JupyterLab.

This tutorial gives you an exhaustive knowledge on Project Jupyter. By the end of this tutorial, you will be able to apply its concepts into your software coding.

Audience

Jupyter notebook is a defacto standard interactive environment for data scientists today. This tutorial is useful for everyone who wants to learn and practice data science libraries of Python/R etc.

Prerequisites

This is not a tutorial to teach Python programming. It describes use of powerful interactive environment to use Python. We assume you have a good understanding of the Python programming language.

If you are beginner to Python and other related concepts, we suggest you to pick tutorials based on these, before you start your journey with Jupyter.

Copyright & Disclaimer

© Copyright 2018 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright & Disclaimer	i
Table of Contents.....	ii
IPYTHON	1
1. IPYTHON – INTRODUCTION	2
Features of IPython	2
History and Development	3
2. IPYTHON – INSTALLATION	4
3. IPYTHON – GETTING STARTED.....	5
Starting IPython from Command Prompt.....	5
Start IPython from conda prompt	6
Magic Functions.....	7
4. IPYTHON — RUNNING AND EDITING PYTHON SCRIPT	8
Run Command	8
Edit Command	8
5. IPYTHON — HISTORY COMMAND	9
6. IPYTHON — SYSTEM COMMANDS.....	10
7. IPYTHON — COMMAND LINE OPTIONS.....	12
Invoking IPython Program	12
Subcommands and Parameters	12
8. IPYTHON — DYNAMIC OBJECT INTROSPECTION.....	14

9.	IPYTHON — IO CACHING	15
10.	IPYTHON — SETTING IPYTHON AS DEFAULT PYTHON ENVIRONMENT	18
11.	IPYTHON — IMPORTING PYTHON SHELL CODE.....	19
12.	IPYTHON — EMBEDDING IPYTHON.....	21
13.	IPYTHON — MAGIC COMMANDS.....	23
	Types of Magic Commands	23
	IPython Custom Line Magic function.....	30
	JUPYTER.....	31
14.	PROJECT JUPYTER — OVERVIEW.....	32
15.	JUPYTER NOTEBOOK — INTRODUCTION.....	33
16.	JUPYTER NOTEBOOK — WORKING WITH JUPYTER ONLINE.....	34
17.	JUPYTER NOTEBOOK — INSTALLATION AND GETTING STARTED.....	35
18.	JUPYTER NOTEBOOK — DASHBOARD.....	37
19.	JUPYTER NOTEBOOK — USER INTERFACE	39
20.	JUPYTER NOTEBOOK — TYPES OF CELLS	42
21.	JUPYTER NOTEBOOK — EDITING.....	43
22.	JUPYTER NOTEBOOK — MARKDOWN CELLS	45
23.	JUPYTER NOTEBOOK — CELL MAGIC FUNCTIONS.....	49
24.	JUPYTER NOTEBOOK — PLOTTING.....	51
25.	JUPYTER NOTEBOOK — CONVERTING NOTEBOOKS.....	53
26.	JUPYTER NOTEBOOK — IPYWIDGETS	54

QTCONSOLE.....	56
27. JUPYTER QTCONSOLE — GETTING STARTED	57
Overview	57
Installation	57
28. JUPYTER QTCONSOLE — MULTILINE EDITING	59
29. JUPYTER QTCONSOLE — INLINE GRAPHICS.....	60
30. JUPYTER QTCONSOLE — SAVE TO HTML.....	61
31. JUPYTER QTCONSOLE — MULTIPLE CONSOLES.....	62
32. JUPYTER QTCONSOLE — CONNECTING TO JUPYTER NOTEBOOK	63
33. SHARING JUPYTER NOTEBOOK — USING GITHUB AND NBVIEWER.....	64
JUPYTERLAB	67
34. JUPYTERLAB— OVERVIEW	68
35. JUPYTERLAB— INSTALLATION AND GETTING STARTED.....	69
36. JUPYTERLAB— INTERFACE	71
37. JUPYTERLAB— INSTALLING R KERNEL	74

IPython

1. IPython – Introduction

Project Jupyter is a suite of software products used in interactive computing. IPython was originally developed by Fernando Perez in 2001 as an enhanced Python interpreter. A web based interface to IPython terminal in the form of IPython notebook was introduced in 2011. In 2014, Project Jupyter started as a spin-off project from IPython.

- Packages under Jupyter project include:
- Jupyter notebook : A web based interface to programming environments of Python, Julia, R and many others
- QtConsole: Qt based terminal for Jupyter kernels similar to IPython
- nbviewer : Facility to share Jupyter notebooks
- JupyterLab : Modern web based integrated interface for all products.

Standard distribution of Python comes with a **REPL (Read-Evaluate-Print Loop)** environment in the form of Python shell with `>>>` prompt. IPython (stands for Interactive Python) is an enhanced interactive environment for Python with many functionalities compared to the standard Python shell.

Features of IPython

IPython offers more features compared to the standard Python. They are as follows:

- Offers a powerful interactive Python shell.
- Acts as a main kernel for Jupyter notebook and other front end tools of Project Jupyter.
- Possesses object introspection ability. Introspection is the ability to check properties of an object during runtime.
- Syntax highlighting.
- Stores the history of interactions.
- Tab completion of keywords, variables and function names.
- Magic command system useful for controlling Python environment and performing OS tasks.
- Ability to be embedded in other Python programs.
- Provides access to Python debugger.

History and Development

IPython was originally developed by Fernando Perez in 2001. Its current version is IPython7.0.1 which requires Python 3.4 version or higher. IPython 6.0 was the first version to support Python 3. Users having Python 2.7 should work with IPython's version 2.0 to 5.7

The concept of computational notebooks started in 80s decade when MATLAB and Mathematica were released. These GUI frontends to the interactive shell had features like text formatting, adding graphics, table and adding mathematical symbols. Sage notebook is also a web based notebook.

Creators of IPython started working on notebook interface for IPython shell in 2005. IPython notebook soon added support of other languages like R and Julia. It was in 2014, that Perez started Jupyter project as a spin-off project from IPython, since IPython project was becoming big with products like notebook server and Qt console added to it.

Since IPython 4.0, all additional components were shifted to Project Jupyter and adding support of other languages to IPython notebook. IPython continues to focus on improvement of its enhanced interpreter feature. It also provides primary kernel to Jupyter notebook frontend.

2. IPython – Installation

IPython is included by default in Anaconda distribution of Python. It can be downloaded from Anaconda's download page <https://www.anaconda.com/download/>. Binaries for all major OS (Windows, MacOS and Linux) and architecture (32 bit and 64 bit) are available on this link.

To install IPython separately in standard Python installation, you can use pip command as shown below:

```
pip3 install ipython
```

IPython internally uses following packages:

IPython dependencies	Functionality
colorama	Cross-platform API for printing colored terminal text from Python
jedi	An autocompletion tool for Python
pickleshare	Small 'shelve' like datastore with concurrency support
prompt_toolkit	Library for building powerful interactive command lines in Python
pygments	Syntax highlighting package written in Python
simplegeneric	Simple generic functions
traitlets	Configuration system for Python applications.

In general, all dependencies get installed automatically. Else, you can install them individually using pip.

3. IPython – Getting Started

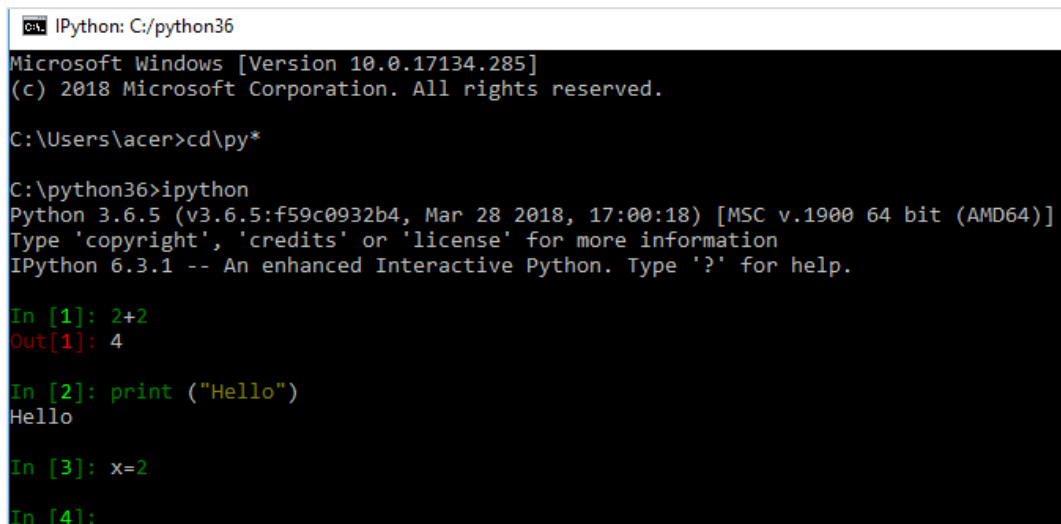
This chapter will explain how to get started with working on IPython.

Starting IPython from Command Prompt.

Before proceeding to understand about IPython in depth, note that instead of the regular `>>>`, you will notice two major Python prompts as explained below:

- **In[1]** appears before any input expression.
- **Out[1]** appears before the Output appears.

Besides, the numbers in the square brackets are incremented automatically. Observe the following screenshot for a better understanding:



```
IPython: C:/python36
Microsoft Windows [Version 10.0.17134.285]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\acer>cd\py*

C:\python36>ipython
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.3.1 -- An enhanced Interactive Python. Type '?' for help.

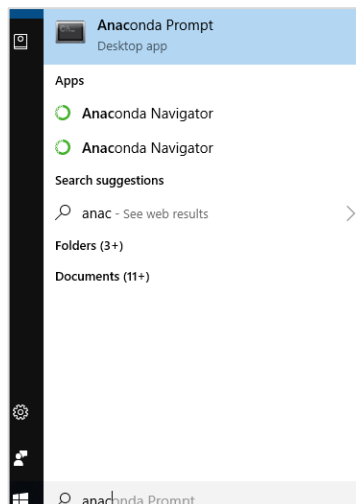
In [1]: 2+2
Out[1]: 4

In [2]: print ("Hello")
Hello

In [3]: x=2

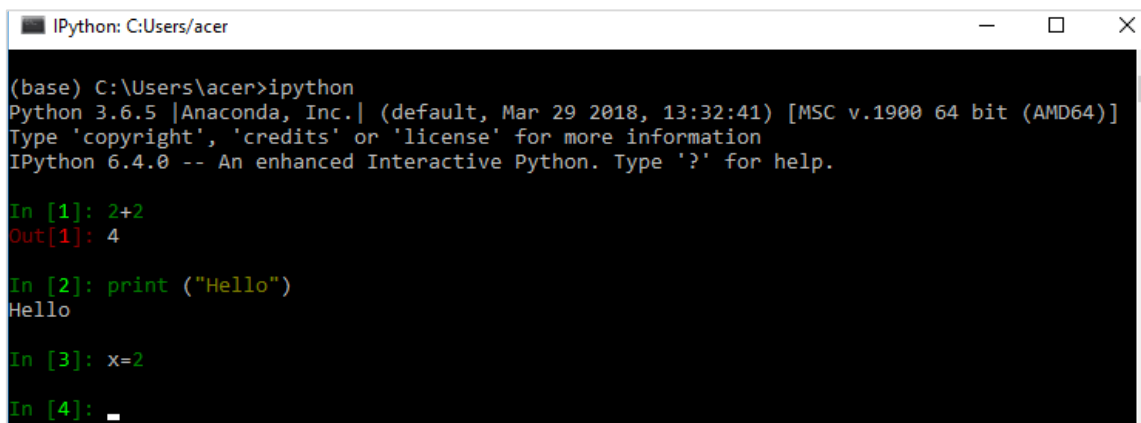
In [4]:
```

Now, if you have installed Anaconda distribution of Python, open Anaconda prompt from start menu



Start IPython from conda prompt

When compared to regular Python console, we can notice a difference. The IPython shell shows syntax highlighting by using different colour scheme for different elements like expression, function, variable etc.



```

IPython: C:\Users\acer
(base) C:\Users\acer>ipython
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: 2+2
Out[1]: 4

In [2]: print("Hello")
Hello

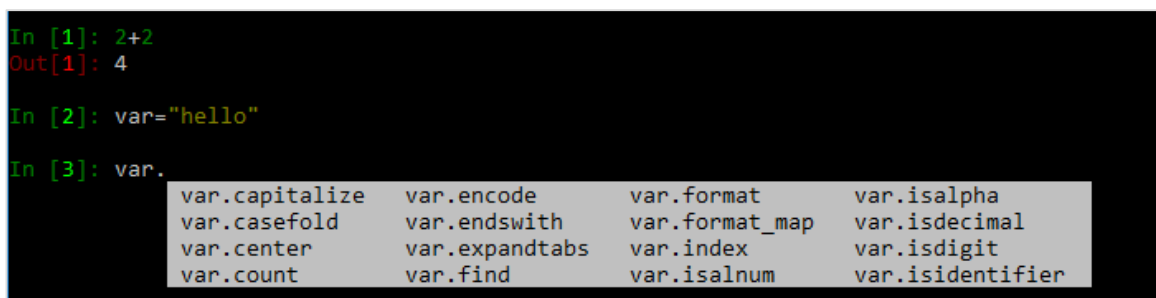
In [3]: x=2

In [4]: _

```

Another useful enhancement is tab completion. We know that each object has one or more methods available as defined in its class. IPython pops up appropriate list of methods as you press tab key after dot in front of object.

In the following example, a string is defined. As a response, the methods of string class are shown.



```

In [1]: 2+2
Out[1]: 4

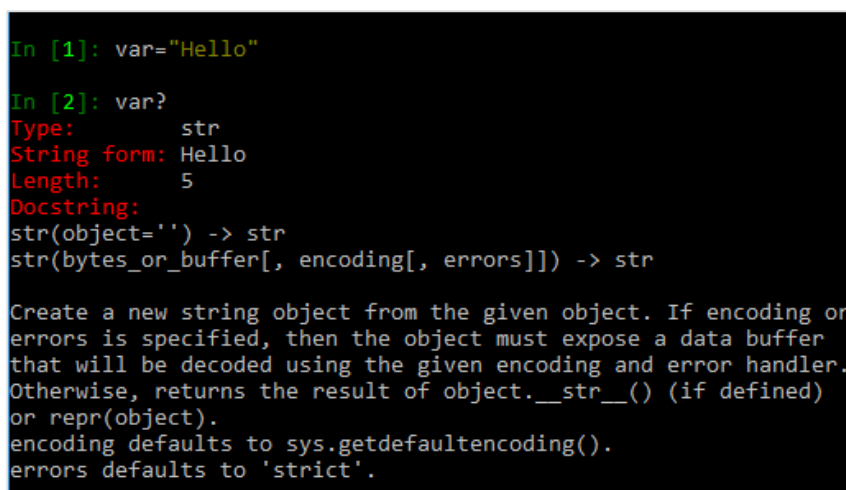
In [2]: var="hello"

In [3]: var.

```

var.capitalize	var.encode	var.format	var.isalpha
var.casefold	var.endswith	var.format_map	var.isdecimal
var.center	var.expandtabs	var.index	var.isdigit
var.count	var.find	var.isalnum	var.isidentifier

IPython provides information of any object by putting '?' in front of it. It includes docstring, function definitions and constructor details of class. For example to explore the string object var defined above, in the input prompt enter **var?**. The result will show all information about it. Observe the screenshot given below for a better understanding:



```

In [1]: var="Hello"

In [2]: var?
Type:      str
String form: Hello
Length:    5
Docstring:
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.

```

Magic Functions

IPython's in-built magic functions are extremely powerful. There are two types of magic functions.

- **Line magics**, which work very much like DOS commands.
- **Cell magics**, which work on multiple lines of code.

We shall learn about line magic functions and cell magic functions in detail in subsequent chapters.

4. IPython — Running and Editing Python Script

In this chapter, let us understand how to run and edit a Python script.

Run Command

You can use **run** command in the input prompt to run a Python script. The run command is actually line magic command and should actually be written as **%run**. However, the **%automagic** mode is always on by default, so you can omit this.

```
In [1]: run hello.py
Hello IPython
```

Edit Command

IPython also provides edit magic command. It invokes default editor of the operating system. You can open it through Windows Notepad editor and the script can be edited. Once you close it after saving its input, the output of modified script will be displayed.

```
In [2]: edit hello.py
Editing... done. Executing edited code...
Hello IPython
welcome to interactive computing
```

Note that hello.py initially contained only one statement and after editing one more statement was added. If no file name is given to edit command, a temporary file is created. Observe the following code that shows the same.

```
In [7]: edit
IPython will make a temporary file named:
C:\Users\acer\AppData\Local\Temp\ipython_edit_4aa4vx8f\ipython_edit_t7i6s_er.py
Editing... done. Executing edited code...
magic of IPython
Out[7]: 'print ("magic of IPython")'
```

5. IPython — History Command

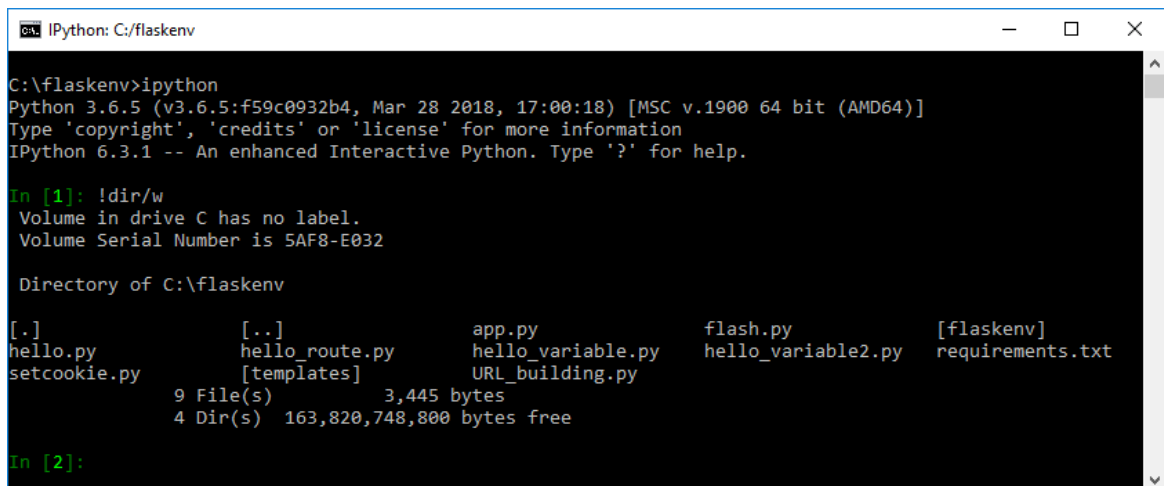
IPython preserves both the commands and their results of the current session. We can scroll through the previous commands by pressing the up and down keys.

Besides, last three objects of output are stored in special variables `_`, `__` and `___`. The **history** magic command shows previous commands in current session as shown in the screenshot given below:

```
In [1]: x=2
In [2]: y=2*x
In [3]: x+y
Out[3]: 6
In [4]: run hello.py
Hello IPython
welcome to interactive computing
In [5]: history
x=2
y=2*x
x+y
run hello.py
history
```

6. IPython — System Commands

If the statement in the input cell starts with the exclamation symbol (!), it is treated as a system command for underlying operating system. For example, **!ls** (for linux) and **!dir** (for windows) displays the contents of current directory



```
C:\flaskenv>ipython
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.3.1 -- An enhanced Interactive Python. Type '?' for help.

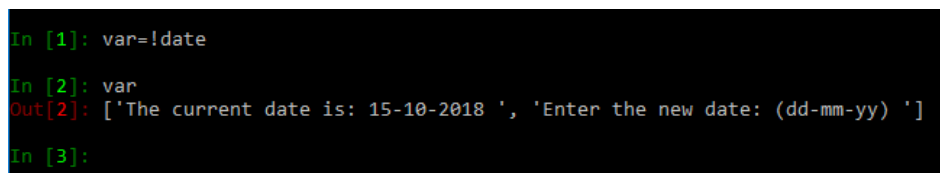
In [1]: !dir/w
Volume in drive C has no label.
Volume Serial Number is 5AF8-E032

Directory of C:\flaskenv

[.]                [..]                app.py              flash.py            [flaskenv]
hello.py           hello_route.py      hello_variable.py   hello_variable2.py requirements.txt
setcookie.py       [templates]         URL_building.py
                   9 File(s)           3,445 bytes
                   4 Dir(s)  163,820,748,800 bytes free

In [2]:
```

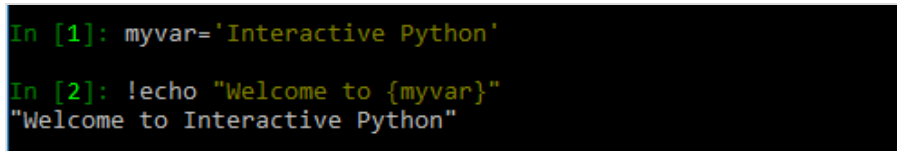
The output of system command can also be assigned to a Python variable as shown below:



```
In [1]: var=!date
In [2]: var
Out[2]: ['The current date is: 15-10-2018 ', 'Enter the new date: (dd-mm-yy) ']
In [3]:
```

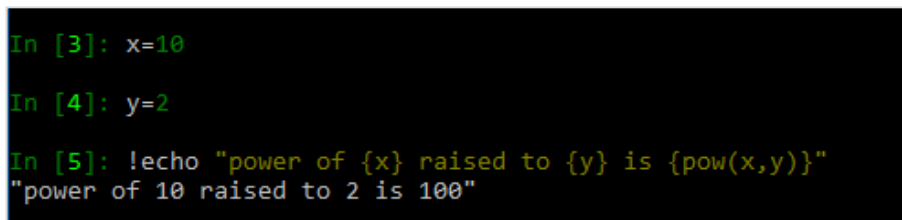
The variable stores output without colors and splits at newline characters.

It is also possible to combine Python variables or expressions with system command calls. Variable in curly brackets {} can be embedded in command text. Observe the following example:



```
In [1]: myvar='Interactive Python'
In [2]: !echo "Welcome to {myvar}"
"Welcome to Interactive Python"
```

Here is another example to understand that prefixing Python variable with \$ also achieves the same result.



```
In [3]: x=10
In [4]: y=2
In [5]: !echo "power of {x} raised to {y} is {pow(x,y)}"
"power of 10 raised to 2 is 100"
```

```
In [6]: z=pow(x,y)
```

```
In [7]: !echo $z  
100
```


7. IPython — Command Line Options

In this chapter, let us understand how to work with various command line options in IPython.

Invoking IPython Program

You can invoke an IPython program using the following options:

```
C:\python36> ipython [subcommand] [options] [-c cmd | -m mod | file] [--] [arg]
```

The file option is a Python script with .py extension. If no other option is given, the script is executed and command prompt reappears.

```
C:\python36>ipython hello.py
Hello IPython
welcome to interactive computing
```

Subcommands and Parameters

An IPython command accepts the following subcommand options:

- **Profile:** Create and manage IPython profiles.
- **Kernel:** Start a kernel without an attached frontend.
- **Locate:** Print the path to the IPython dir.
- **History:** Manage the IPython history database.

An IPython profile subcommand accepts the following parameters:

- **ipython profile create myprofile:** Creates a new profile.
- **ipython profile list:** Lists all available profiles.
- **ipython locate profile myprofile:** Locates required profile.

To install new IPython kernel, use the following command:

```
Ipython kernel -install -name
```

To print the path to the IPython dir, use the following command:

```
C:\python36>ipython locate myprofile
C:\Users\acer\.ipython
```

Besides, we know that:

- The **history** subcommand manages IPython history database.
- The **trim** option reduces the IPython history database to the last 1000 entries.
- The **clear** option deletes all entries.

Some of the other important command line options of IPython are listed below:

--automagic	Turn on the auto calling of magic commands.
--pdb	Enable auto calling the pdb debugger after every exception.
--pylab	Pre-load matplotlib and numpy for interactive use with the default matplotlib backend.
--matplotlib	Configure matplotlib for interactive use with the default matplotlib backend.
--gui=options	Enable GUI event loop integration with any of ('glut', 'gtk', 'gtk2', 'gtk3', 'osx', 'pyglet', 'qt', 'qt4', 'qt5', 'tk', 'wx', 'gtk2', 'qt4').

The sample usage of some of the IPython command line options are shown in following table:

ipython --matplotlib	enable matplotlib integration
ipython --matplotlib=qt	enable matplotlib integration with qt4 backend
ipython --profile=myprofile	start with profile foo
ipython profile create myprofile	create profile foo w/ default config files
ipython help profile	show the help for the profile subcmd
ipython locate	print the path to the IPython directory
ipython locate profile myprofile	print the path to the directory for profile `myprofile`

8. IPython — Dynamic Object Introspection

IPython has different ways of obtaining information about Python objects dynamically. In this chapter, let us learn the ways of dynamic object introspection in IPython.

Use of `?` and `??` provides specific and more detailed information about the object. In the first example discussed below, a simple integer object `a` is created. Its information can be procured by typing a `?` in the input cell.

```
In [1]: a=10

In [2]: a?
Type:      int
String form: 10
Docstring:
int(x=0) -> integer
int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments
are given. If x is a number, return x.__int__(). For floating point
numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string,
bytes, or bytearray instance representing an integer literal in the
given base. The literal can be preceded by '+' or '-' and be surrounded
by whitespace. The base defaults to 10. Valid bases are 0 and 2-36.
Base 0 means to interpret the base from the string as an integer literal.
>>> int('0b100', base=0)
4

In [3]:
```

In the second example, let us define a function and introspect this function object with `?` and `??`.

```
In [4]: def hello():
...:     print ("Hello")
...:

In [5]:

In [5]: hello?
Signature: hello()
Docstring: <no docstring>
File:      c:\users\acer\
```

Note that the magic function `%psearch` is equivalent to the use of `?` or `??` for fetching object information.

End of ebook preview

If you liked what you saw...

Buy it from our store @ <https://store.tutorialspoint.com>