Python Reference (The Right Way) Documentation

Release 0.1

Jakub Przywóski

Sep 30, 2017

Contents

CHAPTER 1

Contents

Introduction

The goal of this documentation is to provide Python Community with high quality lookup reference.

Notes

Update 01/06/2015.

This project is put on the back-burner now. However, I aim to finish uploading the materials sometime this year.

Update 18/01/2015.

Moving all the contents from word files to Sphinx project has proven to be more time consuming than I originally thought. Getting the ver. 1.0 ready will take weeks.

Update

Moving stuff from Word files into reStructuredText is tedious. This is work in progress as of January 2015.

Currently all the material resides on my PC in a form of Word documents. I am going to convert those word documents into .rst files over the course of the next few days. Or weeks possibly if my motivation falters.

And I'm talking about 300-400 A4 pages describing all the Python features with working code examples. It took me about 4 months to put it all together.

All the work was done between April and August 2014.

Scope

Everything here is intended for Python 2.7.X. The reason is simple - this is the version I personally use and its specification is frozen (no new features will be added), so the content is bound to be up to date for good. Moreover, Python 3.X is not catching up - there's like seven or eight people using it worldwide.

This work is not meant to be a total replacement for Python Manuals. As a matter of fact most of the definitions here are based on the official docs. I decided to only cover the core Python's syntax, that means, the stuff that does not require using "import" statement.

This reference is designed to minimize the amount of time needed to look things up. The whole layout is well structured and consistent. I put a lot of emphasis on working code examples and very simple definitions. Being realistic, no one wants to read lengthy passages about some obscure functions, most people only need to glance at the code examples and then copy, paste and modify.

Python Standard Library is beyond the scope of this reference. If you are looking for a description of library modules have a look at:

- Python Module of The Week by Doug Hellman
- Python Reference by Fredrik Lundh (this one is a bit dated, but still top-notch in terms of clarity)
- Official Python Standard Library documentation (terse and lacks examples)

Rationale

Python is such a well-designed, clean and enjoyable to code in language so it sure deserves to have a decent syntax reference. I've been coding in Python for a few months now and whenever I need to check something about syntax 99.9% of the time I end up either on Stackoverflow or some other on-line resource. My main gripes with official docs are too terse descriptions and virtual lack of any code examples as well as lack of any coherent logical structure. It does not have to be this way. Just take a look on Mozilla's JavaScript reference or Microsoft's any .net language or VBA/VBScript references. Those are excellent examples of good technical writing.

Absurdly enough the whole Python documentation stands in contrast to the Zen of Python.

My first idea was to identify main use case scenarios for using any language reference. Luckily there are only two I can think of:

1. I know what I am looking for and I only need a quick refresher on syntax or code snippet to copy/paste end edit for my needs.
2. I want to see if what I need to do has already been implemented (good example is **enum()** function – lots of people implement that pattern themselves). In this case I need to be quickly able to scan through a list of descriptions gathered in one place.

In both cases Python docs fail miserably.

So I decided to introduce the following template logical structures:

Use case 1

This one is used to explain usage of functions/methods. It quickly gives you the info about:

- what does the function do
- what are the inputs
- what is the output

Name

[quick description field – preferably up to 80 characters long]

Syntax

[detailed description of calling this function]

Return Value

[if applicable]

Time Complexity

[if applicable]

Remarks

[further discussion]

Examples

[simple code snippets to illustrate basic usage; the simpler the better]

See Also

[links to related topics]

Use case 2

Used as a list of thematically grouped functions/methods. I decided to organize things by function rather than alphabetically. That's the same way a handyman organizes his tools in the toolbox. Makes needed things easier to find.

Group

Method_a (hyperlink)

[quick description field – preferably up to 80 characters long]

Method_b (hyperlink)

[quick description field – preferably up to 80 characters long]

Definitions

Coding Guidelines

Minimalism

The Zen of Python

PEP 8

Fundamental Data Types

None

None Object that denotes the lack of value.

Numbers

bool **True** and **False** Boolean values. Evaluate to 1 and 0 respectively.

int Integer numbers.

long Long integer numbers.

float Floating point numbers.

complex Complex numbers.

Sequences

str Strings of characters. Immutable.

'unicode'_ Unicode strings of characters. Immutable.

list Indexed list of objects. Mutable.

tuple Indexed list of objects. Immutable.

Mappings

dict Hash table for storing unordered key-value pairs. Mutable.

Sets

set Unordered list of unique objects. Mutable.

Files

file File objects.

Built-In Functions

Functional Programming

map Applies function to every item of an iterable object and returns a list of the results.

filter Returns a sequence from those elements of iterable for which function returns True.

reduce Applies function of two arguments cumulatively to the items of iterable, from left to right, so as to reduce the iterable to a single value.

Numeric Types Conversions and Constructors

bool Returns an expression converted into a Boolean.

int Returns an expression converted into an integer number.

long Returns an expression converted into a long integer number.

float Returns an expression converted into a floating point number.

complex Returns an expression converted into a complex number.

Numeric Types Conversions

bin Returns an integer converted into a binary string.

oct Returns an integer converted into an octal string.

hex Returns an integer converted into a hexadecimal string.

Arithmetic

abs Returns the absolute value of a number.

pow Returns a number raised to a power; or optionally a modulus of the number raised to a power and another number.

round Returns a floating point number rounded to a specified number of decimal places.

divmod Returns quotient and remainder after a division of two numbers.

String Conversions

chr Returns a string of one character whose ASCII code is the specified number.

ord Returns an integer representing the code of the character.

unichr Returns a Unicode character specified by the code.

format Returns a formatted string.

repr Returns a string containing a printable representation of an object.

Sequences Constructors

str Returns a string containing a printable representation of an object.

unicode Returns the Unicode string version of object.

list Converts an object into a list.

tuple Returns a tuple built from iterable.

bytearray Returns a new array of bytes.

buffer Returns a new buffer object which references the object argument.

memoryview Returns a memoryview object.

range Returns a list of arithmetic progressions.

xrange Returns an xrange object.

Mappings Constructors

dict Returns a dictionary object.

set Returns a set type initialized from iterable.

frozenset Returns a frozenset object.

Operating on Containers

enumerate Returns an enumerate object.

len Returns an int type specifying number of elements in the collection.

reversed Returns a reverse iterator over a sequence.

sorted Returns a sorted list from the iterable.

sum Returns a total of the items contained in the iterable object.

zip Returns a list of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables.

slice Returns a slice object.

Iterators

iter Returns an iterator object.

next Retrieves the next item from the iterator by calling its next() method.

Comparisons

cmp Compares two objects and returns an integer according to the outcome.

max Returns the largest item in an iterable or the largest of two or more arguments.

min Returns the smallest item from a collection.

all Returns a Boolean value that indicates whether the collection contains only values that evaluate to True.

any Returns a Boolean value that indicates whether the collection contains any values that evaluate to True.

Identity

hash Return the hash value of the object (if it has one).

id Returns the "identity" of an object.

File Objects Constructors

file Returns a file object.

open Opens a file returning a file object.

Object Oriented Functions

classmethod Returns a class method for the function.

property Returns a property attribute for new-style classes (classes that derive from object).

staticmethod Returns a static method for function.

super Returns a proxy object that delegates method calls to a parent or sibling class of type.

setattr Assigns a value to the object's attribute given its name.

getattr Returns the value of the named attribute of object.

delattr Deletes the named attribute of an object.

hasattr Returns a Boolean stating whether the object has the specified attribute.

isinstance Returns a Boolean stating whether the object is an instance or subclass of another object.

issubclass Returns a Bool type indicating whether an object is a subclass of a class.

vars Returns the mapping of an object's (writable) attributes.

dir Returns the list of names in the current local scope. If supplied with an argument attempts to return a list of valid attributes for that object.

type (1) Returns the type of an object (constructor name).

type (2) Returns a new type object.

Information

callable Returns a Boolean stating whether the object argument appears callable.

globals Returns a dictionary representing the current global symbol table.

locals Returns a dictionary representing the current local symbol table.

help Invokes the built-in help system.

System

__import__ Imports a module.

reload Reloads a previously imported module.

compile Returns an AST or code object.

execfile Evaluates contents of a file.

eval Returns a result of the evaluation of a Python expression.

input Evaluates user input.

intern Enters the string into interned strings table (if not already there).

print Returns a printed representation of the objects.

raw_input Reads a line from standard input stream.

Misc

object Returns a new featureless object.

apply Returns the result of a function or class object called with supplied arguments.

basestring This abstract type is the superclass for str and unicode. It cannot be called or instantiated, but it can be used to test whether an object is an instance of str or unicode.

coerce Returns a tuple consisting of the two numeric arguments converted to a common type.

Comprehensions and Generator Expression

Comprehensions

[] list comprehension Returns a list based on existing iterables.

{} set comprehension Returns a set based on existing iterables.

{} dictionary comprehension Returns a dictionary based on existing iterables.

Generator Expression

() generator expression Returns an iterator over elements created by using list comprehension.

Container Data Access

Brackets Operators

[] (indexing) Gives access to a sequence's element.

[] (slicing) Gives access to a specified range of sequence's elements.

[] (dict key lookup) Returns the value associated with the given key.

[] (ellipsis) Gives access to a specified range of array's elements.

Operators

Arithmetic Operators

+ (addition) Returns the sum of two expressions.

- (subtraction) Returns the difference of two expressions.

*** (multiplication)** Returns the product of two expressions.

**** (power)** Returns the value of a numeric expression raised to a specified power.

/ (division) Returns the quotient of two expressions.

// (floor division) Returns the integral part of the quotient.

% (modulus) Returns the decimal part (remainder) of the quotient.

Assignment Operators

= (simple assignment) Assigns a value to a variable(s).

+= (increment assignment) Adds a value and the variable and assigns the result to that variable.

-= (decrement assignment) Subtracts a value from the variable and assigns the result to that variable.

***= (multiplication assignment)** Multiplies the variable by a value and assigns the result to that variable.

/= (division assignment) Divides the variable by a value and assigns the result to that variable.

****= (power assignment)** Raises the variable to a specified power and assigns the result to the variable.

%= (modulus assignment) Computes the modulus of the variable and a value and assigns the result to that variable.

//= (floor division assignment) Floor divides the variable by a value and assigns the result to that variable.

Relational Operators

== (equal) Returns a Boolean stating whether two expressions are equal.

!= (not equal) Returns a Boolean stating whether two expressions are not equal.

> (greater than) Returns a Boolean stating whether one expression is greater than the other.

>= (greater than or equal) Returns a Boolean stating whether one expression is greater than or equal the other.

< (less than) Returns a Boolean stating whether one expression is less than the other.

<= (less than or equal) Returns a Boolean stating whether one expression is less than or equal the other.

Boolean Operators

and Returns the first operand that evaluates to *False* or the last one if all are *True*.

or Returns the first operand that evaluates to *True* or the last one if all are *False*.

not Returns a boolean that is the reverse of the logical state of an expression.

Conditional Operator

if else Returns either value depending on the result of a Boolean expression.

Identity

is Returns a Boolean stating whether two objects are the same.

Membership

in Returns a Boolean stating whether the object is in the container.

Deletion

'del'_ Removes object.

Callables Operators

*** (tuple packing)** Packs the consecutive function positional arguments into a tuple.

**** (dictionary packing)** Packs the consecutive function keyword arguments into a dictionary.

*** (tuple unpacking)** Unpacks the contents of a tuple into the function call.

**** (dictionary unpacking)** Unpacks the contents of a dictionary into the function call.

@ (decorator) Returns a callable wrapped by another callable.

() (call operator) Calls a callable object with specified arguments.

lambda Returns an anonymous function.

Bitwise Operators

& (bitwise AND) Returns the result of bitwise AND of two integers.

| (bitwise OR) Returns the result of bitwise OR of two integers.

^ (bitwise XOR) Returns the result of bitwise XOR of two integers.

<< (left shift) Shifts the bits of the first operand left by the specified number of bits.

>> (right shift) Shifts the bits of the first operand right by the specified number of bits.

~ (bitwise complement) Sets the 1 bits to 0 and 1 to 0 and then adds 1.

Bitwise Assignment Operators

&= (bitwise AND assignment) Performs bitwise AND and assigns value to the left operand.

|= (bitwise OR assignment) Performs bitwise OR and assigns value to the left operand.

^= (bitwise XOR assignment) Performs bitwise XOR and assigns value to the left operand.

<<= (bitwise right shift assignment) Performs bitwise left shift and assigns value to the left operand.

>>= (bitwise left shift assignment) Performs bitwise right shift and assigns value to the left operand.

Misc

; (statement separator) Separates two statements.

(line continuation) Breaks the line of code allowing for the next line continuation.

. (attribute access) Gives access to an object's attribute.

String and Sequence Operators

+ (concatenation) Returns a concatenation of two sequences.

*** (multiple concatenation)** Returns a sequence self-concatenated specified amount of times.

% (string formatting operator) Formats the string according to the specified format.

Sequence Assignment Operators

+= (concatenation assignment) Concatenates the sequence with the right operand and assigns the result to that sequence.

***= (multiple concatenation assignment)** Multiple concatenates the sequence and assigns the result to that sequence.

Statements

Flow Control

'if'_

'elif'_

'else'_

Loops

for in Loops over elements of an iterable object.

while Executes block of code repeatedly while the specified condition is True.

continue Skips the execution of the code below it and starts a new cycle of the loop.

break Terminates the execution of a loop.

else (2) Executes specified block of code after loop terminating condition other then break was met.

Functions

'def'_

'return'_

'pass'_

Generators

‘yield‘_

Classes

‘class‘_

‘del‘_

Context Managers

‘with‘_

System

‘exec‘_

‘print‘_

Imports and Scope

‘import‘_

‘from‘_

‘as‘_

‘global‘_

Assertions

assert Raises AssertionError if the specified expression evaluates to False.

Exceptions Handling

‘try‘_

‘except‘_

‘finally‘_

‘raise‘_

Other Objects

Data Types

frozenset Unordered list of unique objects. Immutable.

bytearray Sequence of integers in the range between 0 and 255. Mutable.

memoryview View of the object's raw byte data.

Method Decorators

classmethod Method that takes class as its first arguments (instead of a class instance).

staticmethod Method that explicitly does not take the class instance as its first argument.

property Allows for proper use of getter, setter and deleter methods in Python.

Others

function A function object.

generator A generator function object.

code Compiled Python code.

slice Slice objects.

Double Underscore Methods and Variables

Direct Attribute Access

Methods used for direct set, get and delete operations on attributes.

Descriptor Protocol

Used for management over attribute access for class instances.

Comparisons

Containers

Context Managers

Numeric Methods

Object Attributes

Pickle Protocol

Exceptions

Constants

Boilerplate

‘if __name__ == ‘__main__’: main()‘_ Prevents main() from being executed during imports.

‘#!/usr/bin/env/python‘_ UNIX specific.

‘#!/usr/local/bin/python‘_ UNIX specific.

‘#!/usr/bin/python‘_ UNIX specific.

‘# -*- coding: utf-8 -*-‘_ Declares usage of UTF-8 characters in the script.

Glimpse of the PSL

Data Structures and Algorithms

‘array‘_

‘bisect‘_

‘heapq‘_

‘Collections Counter‘_

‘Collections defaultdict‘_

‘Collections deque‘_

‘Collections namedTuple‘_

‘Collections orderedDict‘_

‘Queue‘_

‘functools‘_

‘itertools‘_

Time

time

Files and Folders

'os'_

'os.path'_

'shutil'_

'glob'_

'ZipFile'_

Resources

#TODO Books, on-line courses on Python etc.

Licence

CHAPTER 2

Indices and tables

- genindex
- modindex
- search