Adventures in Python Documentation

Release 0.0.1

Ceasar Bautista

Jun 23, 2017

Contents

This text provides an introduction to the Python language.

Contents:

CHAPTER 1

Introduction

Before studying Python, the first thing we have to consider is how do we know if we have reached our goal. That is, how will we know if we understand Python?

Well, what does it mean to understand anything?

According to Aristotle:

> When the objects of an inquiry, in any department, have principles, conditions, or elements, it is through acquaintance with these that knowledge, that is to say scientific knowledge, is attained. For we do not think that we know a thing until we are acquainted with its primary conditions or first principles, and have carried our analysis as far as its simplest elements. - Aristotle, Physics

So, to understand Python is to understand each of its parts. The process is of course called analysis.

The easiest and natural way to perform analysis is to start from the things we know and which are obvious to us and proceed toward those which are more knowable by nature.

So, to analyze Python, I think it makes sense to first analyze its superclasses.

What is a programming language? A programming language is a formal language designed for communicating instructions to machine.

What is a formal language? A formal language is a language designed for use in situations in which natural language is unsuitable.

What is a language? A language is a system of communication for encoding and decoding information.

The main point of all of this is to demonstrate that to study Python alone would be only to study a system of communication, and not how to communicate. To make an analogy to chess, studying Python that way would be like studying how to play chess (that is, the rules) without studying how to play _well_.

"Well" is, of course, subjective, which means there is an art to both chess and Python. The name for this art, the art of discourse, is of course "rhetoric".

Note, we will not study logic, which is the art of thinking (and the third part of the trivium). Rather, we will consider logic a prerequisite.

So to conclude: This goal of this text is to ensure you (1) understand how to symbolize thought in Python (the grammar and lexicon) and (2) how to use Python to communicate thought well.

The study of a programming language can roughly divided into two parts:

1. Language (how to write correctly)
2. Rhetoric (how to write well)

Language

A language is a system of communication.

Analysis

All languages consist of two parts:

1. Syntax
2. Lexicon

Syntax

The syntax of a language defines what makes a sentence valid and what makes a sentence invalid.

Example: "I go to the store" is valid. Example: "I goes to the store" is invalid. Example: "Store I go to the" is invalid.

The syntax of a language consists of:

1. Morphology
2. Grammar

Lexicon

The lexicon of a language enumerates the words in the language.

Example: "store" is a word. Example: "jabberwocky" is a not word.

Rhetoric

Rhetoric is the art of discourse (or speech and writing).

Rhetoric consists of five parts:

1. Invention - How does one come up with what to write?
2. Arrangement - How does one arrange a text?
3. Style - How
4. Memory
5. Delivery

Invention correspond to architecture. Arrangement corresponds to design. Style is conventional and is addressed by style guides. Memory has no correspondence. (It was an artifact of paper being hard to come by. Arguably simplicity?) Delivery has no correspondence. (Arguably this goes down to things like medium, but there is always one).

Synthesis

The grammar of a language consists of syntax

Morphology in programming exists basically as maps with one arguments that may have optional arguments.

Example: *reversed(nums)* means "Numbers in reverse" which could

The lexicon of a programming language consists of:

1. Built-ins
2. The standard library
3. Packages

—

All communication is rhetorical.

CHAPTER 2

Grammar

Glyphs

A glyph is an element of writing: an individual mark.

Python uses glyphs as a shorthand notation for common data structures:

```
[1, 2] # list
(1, 2) # tuple
{1, 2} # set
{'a': 1, 'b': 2} # dictionary
```

Bad Parts

The syntax of tuple glyphs is inconsistent:

```
a = (1, 2,)

assert isinstance(a, tuple)

b = (1, 2)

assert isinstance(b, tuple)

c = (1,)

assert isinstance(c, tuple)

d = (1)

assert isinstance(d, int)

# e = (,)
```

```
# Invalid syntax

f = ()

assert isinstance(f, tuple)
```

Note: This has existed since at least 1991. See: http://www.python.org/search/hypermail/python-1992/0278.html

The glyphs for sets and dictionaries are potentially ambiguous:

```
a, b = {1: 'a', 2: 'b'}, {1, 2}
c, d = {1: 'a'}, {1}
e, f = {}, {} # are these dictionaries or sets?
```

The glyphs for are list comprehensions, set comprehensions, and generator expressions are inconsistent– one would expect tuple comprehensions as well:

```
>>> [x for x in range(5)]
[0, 1, 2, 3, 4]
>>> {x for x in range(5)}
set([0, 1, 2, 3, 4])
>>> (x for x in range(5))
<generator object <genexpr> at 0x10698feb0>
```