The Python Tutorial

Lina Liu

Oct, 19, 2022

Why Python

1. Powerful
2. Efficient high-level data structures
3. Simple but effective OOP
4. Ideal scripting language
5. Free third party python modules, tools, programs
6. Easily extended in other languages

What in python Tutorial

1. Basic concepts and noteworthy features
2. Lots of examples
3. Extensions:
    1. Standard objects and modules: python standard library
    2. Language definition: python language reference
    3. Write extensions in C/C++: Extending and Embedding the python interpreter/ Python API reference manual

Other References

1. Description of standard objects and modules
    1. The Python Standard Library
2. Formal definition of the language
    1. The Python Language Reference
3. Extensions in C and C++
    1. Extending and Embedding the Python Interpreter
4. The Glossary is also worth going through
    1. >>> the default python prompt of the interactive shell
    2. Code examples can be executed interactively in the interpreter
5. Books covering python in depth
    1. Effiective python
    2. Python cookbook
    3. Fluent Python

What you will get?

1. Give you a good idea of the languages's flavor and style
2. You will be able to read and write python modules and programs
3. Be ready to the next step
    1. Learn more about the various python library modules described in The python standard library

Contents

Shining Python

1. Automate you task.
    1. Perform a search-and-replace over a large number of text files
    2. Rename and rearrange a bunch of photo files
    3. Write a small custom database
    4. A specialized GUI application
    5. Simple game
2. Compare with C/C++/java, the burdens of C/C++/Java
    1. You may find the usual write/compile/test/re-compile cycle of above languages is too slow
    2. Writing the test code a tedious task
    3. When you use an extension language, we may design/implement a whole new language
3. Compare with Unix Shell, the burdens of Unix Shell
    1. Shell scripts are best at moving around files and changing text data, but
    2. Not well-suited for GUI applications or games.
4. Compare with Awk or Perl
    1. Python offer much more structure and support for large programs.
    2. Python offers much more error checking than C
    3. Python has high-level data types built in.
5. Easily integrated modules
    1. Sprit your programs into modules that can be reused in other python programs
6. Interpreted language
    1. No compilation and linking is necessary.
    2. Interpreter can be used interactively.
7. Tidy and readably language
    1. Express complex operations in a single statement
    2. No beginning and ending brackets
    3. No variable or argument declaration are necessary
8. Python is extensible
    1. Once you are hooked, you can link the python interpreter into an application written in C
9. How the python name come from
    1. Named after the BBC show. “Monty Python’s Flying Circus”

Python Interpreter

1. Where python interpreter located
    1. Usually installed as /usr/local/bin/python3.10
    2. Alternative location: /usr/local/python
2. How to start python interpreter
    1. Putting /usr/local/bin in you Unix Shell's search path
3. How to exit the python interpreter
    1. Ctrl-D on Unix
    2. Typing cmd: quit()
4. What python interpreter do
    1. Interactive editing
    2. History substitution
    3. Code completion
    4. Read and execute cads interactively
    5. When called with a file name argument or with a file as standard input, it reads and executes a script from that file
5. How to start python interpreter
    1. $ Python3.10
    2. >>>
6. The Interpreter and its Environment
    1. Python source files are treated as encoded in UTF-8

Python Introduction

1. Numbers
    1. Int / float/ Decimal/ Fraction
2. Strings
    1. Enclosed in single quotes or double quotes
    2. Special characters are escaped with backslashes
    3. print() function produces a more readable output
    4. If you don’t want characters prefaced by \ to be interpreted as special characters, you can use raw string by adding an r before the first quote.
    5. String can be concatenated with + and repeated with *
        1. 3 * “un” + ium
            1. Unununium
        2. Two or more string literals next to each other are automatically contented
        3. Strings can be indexed, with the first character having index 0
            1. Indices may also be negative numbers, to start counting from the right
            2. Slicing is supported, slicing allows you to obtain substring
        4. Strings cannot be changed — they are immutable
        5. Built in function len() returns the length of a string
        6. Strings are examples of sequence types
        7. Strings support a large number of method for basic transformation and searching
3. Lists
    1. Compound different data types, used to group together other values.
    2. List can be written as a list of comma-separated values(items) between square brackets
    3. List can be indexed and sliced
    4. List value can be changes, lists are a mutable type
    5. You can add new items at the end of the list, by using append() method
    6. Assign to slices is also possible, and this can even change the size of the list, or clear it entirely
    7. It’s possible to nest lists(create lists containing other lists)s

Control Flow Tools

1. If statements
    1. If / elif /else
2. For statements
    1. Iterates over the items of any sequence, in the order that they appear in the sequence
    2. Tricky:
        1. Modifies a collection which iterating over the same collection can be tricky to get right
            1. Loop over a copy of the collection or to create a new collection
3. The range() function
    1. range() function for you if you need to iterate over a sequence of numbers
    2. To iterate over the indices of a sequence, you can combine range() and len()
        1. Sometimes enumerate() func is more convenient.
    3. What range() returned?
        1. Returns items of the iterable sequence, but it's not a list
4. Break/continue/else
5. Pass
6. Match statements
    1. Like switch statement in c
    2. Python 3.9 doesn't support match statement!!!!!!!!!!!!!!!!!
7. Defining functions
    1. Default Argument Values
        1. The default value is evaluated only once, this makes a difference when the default is a mutable object such as list, dictionary, or instances of most classes.
8. Keyword arguments
    1. Functions can be called using keyword arguments of the form kwarg=value
    2. Keyword arguments must follow positional arguments
    3. What if **name present?
        1. It means it receives a dictionary containing all keyword arguments except for those corresponding to a formal parameter
        2. *name must occur before **name
        3. Examples are in Python_tutorial_controlflow.py
        4. The order in the keyword arguments is guaranteed to match the order in which they were provided in the function cal
9. Special Parameters

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
      ----------    ----------     ----------
        |             |                  |
        |        Positional or keyword   |
        |                                - Keyword only
         -- Positional only
```

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
```

Control Flow Tools

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
      -----------    ----------     ----------
        |             |                  |
        |        Positional or keyword   |
        |                                - Keyword only
         -- Positional only
```

9. Special parameters
    1. Arguments may be passed to a python function either by position or ex
    2. Rules:
        1. Items are passed by position, by position or keyword, or by keyword
    3. / and * are optional, when used, these symbols indicate the kind of parameter by how the arguments may be passed to the function
        1. Examples are in Python_tutorial_controlflow.py
        2. The names of positional-only parameters can be used in **kwdds without ambiguity.
    4. Recap
        1. Use positional-only if you want the name of the parameters not be available
        2. Use keyword-only when names have meaning and the function definition is more understandable by being explicit with names
        3. For an API, use positional-only to prevent breaking API changes if the parameter's name is modified in the future.
10. Arbitrary Argument Lists
    1. Frequently used option is to specify that a function can be called with an arbitrary number of arguments
        1. Arguments will be wrapped up in a tuple
    2. See examples in Python_tutorial_controlflow.py
11. Unpacking argument lists
    1. This happens when the arguments are already in a list or tuple but need to be unpacked for a function call requiring separate positional argument

    range() functions expects separate start and stop arguments. If they are not available separately, write the function call with *-opeartor to unpack the arguments out of a list or tuple:

    3. Dictionaries can deliver keyword arguments with the **-operator
12. Lambda Expressions
    1. Small anonymous functions can be created with the lambda keyword.
        1. Lambda function returns the sum of its two arguments: lambda a, b: a+b.
        2. Lambda functions can be used wherever function objects are required
        3. Lambda function syntactically restricted to a single expression
        4. Like nested function definitions, lambda functions can reference variables from the containing scope.
        5. Examples use a lambda expression to return a function

Control Flow Tools

13. Documentation Strings
    1. First line always be a short, concise summary of the object's purpose.
    2. First line not explicitly state the objects's name or type, since these are available by other means
    3. The second line should be blank, visually separating the summary from the rest of the description.
    4. Following lines should be one or more paragraphs describing the objects's calling conventions, its side effects, etc.
14. Function Annotations
    1. Optional
    2. Return annotations are defined by a literal ->, followed by an expression

15. Coding Style
    1. PEP 8: https://peps.python.org/pep-0008/
        1. Readable and eye-pleasing coding style.
    2. Use 4-space indentation, and no tabs
        1.
```
def f(ham: str, eggs: str = 'eggs') -> str:
```
    3. Wrap lines so that they don't exceed 79 characters
    4. Blank lines to separate functions and classes, and larger blocks of code inside functions
    5. Comments
    6. Use docstrings.
    7. Use spaces around operators and after commas
    8. Name your classes and functions consistently, the convention is to use UpperCamelCase for classes and lowercase_with_underscores for functions and methods.
    9. Always use self as the name for the first method arguments
10. UTF-8