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

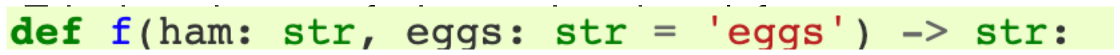
    range() functions expects separate start and stop arguments. If they are not available separately, write the function call with *-opeartor to unpack the arguments out of a list or tuple:

    ```
    args = [3, 6]
    list(range(*args))
    ```

    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. 

```python
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

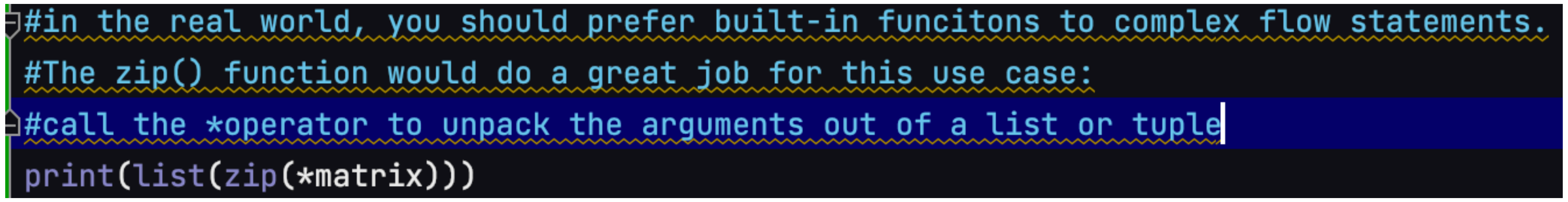
Data Structures

1. Lists
    1. All the methods of list objects
        1. list.append(x)
            1. Add an item to the end of the list
        2. list.extend(iterable)
            1. Extend the list by appending all the items from the iterable.
        3. list.insert(i, x)
            1. Insert an item at a given position.
            2. a.insert(0,x) inserts at the front of the list
            3. a.insert(len(a), x) is equivalent to a.append(x).
        4. list.remove(x)
            1. Remove the first item from the list whose value is equal to x, ValueError if there is no such item
        5. list.pop([i])
            1. Remove the item at the given position in the list, and return it.
            2. If no index is specified, a.pop() removes and returns the last item in the list.
            3. The square brackets around the i in the method signature denote that the parameter is optional, not that you should type square brackets at that position
        6. list.clear()
            1. Remove all items from the list
        7. list.index(x[,start[,end]])
            1. Return zero-based index in the list of the first item whose value is equal to x
        8. list.count(x)
            1. Return the number of times x appears in the list
        9. list.sort(*,key=none,reverse=False)
            1. Sort the items of the list in place
        10. list.reverse()
            1. Reverse the elements of the list in place
        11. list.copy()
            1. Return a shallow copy of the list.
        12. Methods like insert, remove or sort that only modify the list have no return value printed —they return default None. This is design principle for all mutable data structures in Python
        13. Not all data can be sorted or compared, [None, ‘hello’, 10] doesn’t sort because integers can’t be compared to strings and None can’t be compared to other types.

Data Structure

Using lists

1. Using lists as stacks
    1. Very easy to use a list as a stack, where that last element added is the first element retrieved(last in, first out)
    2. To add an item to the top of the stack, use append(), to retrieve an item from the top of the stack, use pop() without an explicit index
2. Using lists as queues
    1. Lists are not efficient for this purpose.
    2. To implement a queue, use collections.deque which was designed to have fast append and pops from both ends
3. List comprehensions
    1. List comprehensions provide a concise way to create lists.
    2. Make new lists where each element is the result of some operations applied to each member of another sequence or iterable
    3. Create a subsequence of elements that satisfy a certain condition
4. Nested List comprehensions
    1. The initial expression in a list comprehension can be any arbitrary expression



```
#in the real world, you should prefer built-in funcitons to complex flow statements.
#The zip() function would do a great job for this use case:
#call the *operator to unpack the arguments out of a list or tuple
print(list(zip(*matrix)))
```

Data Structures

Del statement / Tuples / Sequences

5. The del statement
    1. Used to remove an item from a list given its index instead of its value.
    2. Used to remove slices from a list
    3. Used to remove the entire list
    4. Used to delete entire variables
6. Tuples and Sequences
    1. Sequence data types: list, string, tuple, can do indexing and slicing operations
    2. A tuple consists of a number of values separated by commas
    3. Tuples are immutable, ‘tuple’ object does not support item assignment
        1. It is not possible to assign to the individual items of a tuple
        2. It is possible to create tuples which contain mutable objects, such as lists
        3. Tuples usually contain a heterogeneous sequence of elements that are accessed via unpacking or indexing
    4. Tuples may be input with or without surrounding parentheses
    5. Tricky of construction of tuples containing 0 or 1 items: quirks!
        1. Empty tuples are constructed by an empty pair of parentheses
            1. Empty = ()
        2. One item tuple is constructed by following a value with a comma
            1. Singleton = ‘hello’,
    6. Sequence unpacking
        1. Works for any sequences on the right hand side.
        2. Sequence unpacking requires that there are as many variables on the left side of the equals sign as there are elements in the sequence.
        3. Multiple assignment is just combination of tuple packing and sequence unpacking
        4. `x, y, z = t`

Data Structure

Sets &

7. Sets
    1. A set is an unordered collection with no duplicate elements
    2. Mainly used on membership testing and eliminating duplicate entries
    3. Support mathematic operations:
        1. Union
        2. Intersection
        3. Difference
        4. Symmetric difference
    4. How to create a set?
        1. Curly braces
            1. Note: to create an empty set you have to use set() not {}
                1. {} used to create an empty dictionary
        2. set() function

Data Structure

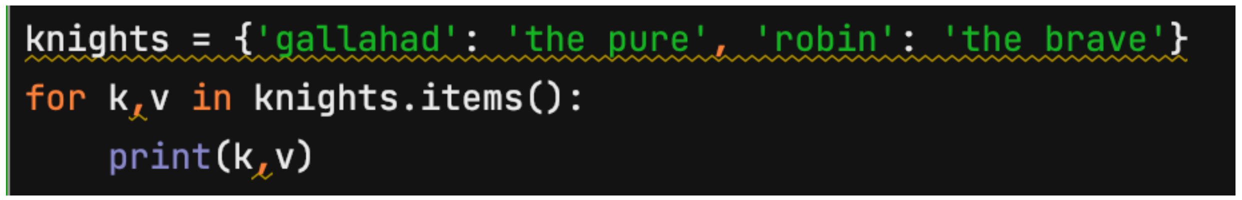
Dictionaries

8. Dictionaries
    1. Dictionaries found in other languages as “associative memories” or “associative arrays”
    2. Dictionaries are indexed by keys
        1. Keys can be any immutable type
        2. Strings and numbers can always be keys
        3. Tuples can be used as keys if they contain only strings, numbers, or tuples
        4. Lists can not be used as keys
            1. Because list can be modified in place using index assignments, slice assignments, or methods like append() and extend()
    3. Dictionary are a set of key: value pairs
        1. Keys are unique
        2. A pair of braces create an empty dictionary: {}
        3. Placing a comma separated list of key:value pairs within the braces
    4. Main operations on a dictionary
        1. Storing a value with some key
        2. Extracting the value given the key
        3. Delete a key: value pair with del
        4. If you store using a key that is already in use, the old value associated with that key is forgotten
        5. Error to extract a value using a non existent key.
        6. list(d)
            1. Return a list of all the keys used in the dictionary
        7. sorted(d)
            1. Return a list of all the keys in sorted order
        8. in keyword
            1. Check whether a single key is in the dictionary
                1. ‘Jacky’ not in tel
        9. dict() constructor builds dictionaries directly from sequences of key-value pairs

Data Structure

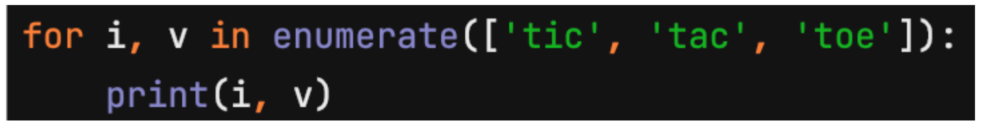
Looping Techniques

9. Looping techniques
    1. Dictionary use items() method to retrieve key and value at the same time



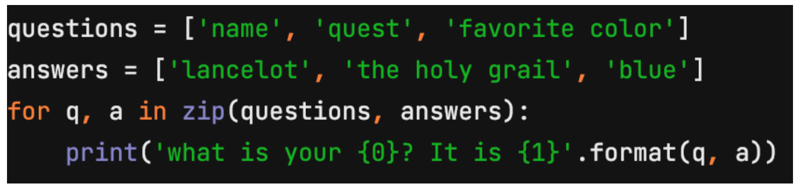
```
knights = {'gallahad': 'the pure', 'robin': 'the brave'}
for k,v in knights.items():
    print(k,v)
```

    2. Sequence using the enumerate() function to retrieve index and value at the same time



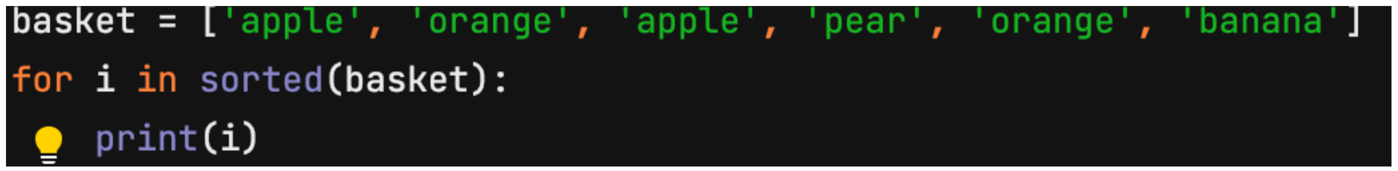
```
for i, v in enumerate(['tic', 'tac', 'toe']):
    print(i, v)
```

    3. Using zip() function to loop over 2 or more sequences at the same time



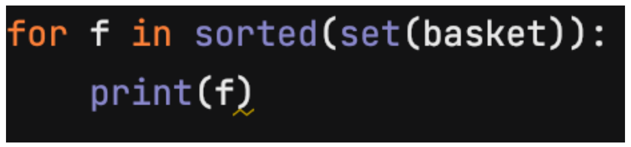
```
questions = ['name', 'quest', 'favorite color']
answers = ['lancelot', 'the holy grail', 'blue']
for q, a in zip(questions, answers):
    print('what is your {0}? It is {1}'.format(q, a))
```

    4. Using sorted() function to loop over a sequence in sorted order, sorted() function returns a new sorted list while leaving the source unaltered.



```
basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
for i in sorted(basket):
    print(i)
```

    5. Using set() combined with sorted() function over a sequence is an idiomatic way to loop over unique elements of the sequence in sorted order.



```
for f in sorted(set(basket)):
    print(f)
```

    6. The list may changed while you are looping over it, it is often simpler and safer to create a new list instead.

Data Structure

More on conditions / Comparing sequences

10. More on conditions
    1. Conditions used in while and if statement can contain any operators
    2. Comparison operator in and not in are membership tests that determine whether a value is in or not in a container
    3. Operator is and is not compare whether two objects are really the same object
    4. Comparisons can be chained
    5. Comparisons may be combined using and and or,
11. Compare sequences

```
(1, 2, 3)              < (1, 2, 4)
[1, 2, 3]              < [1, 2, 4]
'ABC' < 'C' < 'Pascal' < 'Python'
(1, 2, 3, 4)           < (1, 2, 4)
(1, 2)                 < (1, 2, -1)
(1, 2, 3)             == (1.0, 2.0, 3.0)
(1, 2, ('aa', 'ab'))   < (1, 2, ('abc', 'a'), 4)
```