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.
```
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

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

Looping Techniques

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

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

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

    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)
```

Modules

1. Why create modules?
    1. Normally, we create a script to write python code, As your program gets longer, you may want to split it into several files for easier maintenance.
    2. You may also want to use a handy function that you've written in several programs without copying its definition into each program
    3. Python module provide a way to put definitions in a file and use them in a script
2. How to use module?
    1. Definitions from a module can be imported into other modules
3. How to write a module

    Example in module.py

Modules

More on Modules

4. How to import modules?
    1. Import function name from module directly
        1. From fib import fib, fib2
    2. Import all names that a module defines
        1. From fib import *
            1. The practice of importing * from a module or package is frowned upon, since it often causes poorly readable code.
3. as, if the module name is followed by as, then the name following as is bound directly to the imported module.
    1. Import fibo as fib
4. NOTE: each module is only imported once per interpreter session, if you change your modules, you must restart the interpreter, if it's just one module you want to test interactively, use import lib.reload()
    1. Import importlib
    2. Importlib.reload(module name)

Modules

The Module Search Path

5. How interpreter find the module?
    1. Interpreter first search for a built-in module with that name
    2. If not found, interpreter searches for a file named spam.py in a list of directories given by the variable sys.path.
        1. Sys.path is initialized from following locations:
            1. The directory containing the input script
            2. PYTHONPATH
            3. The installation-dependent default(site-package directory, pycharm)

Modules

Compiled python files

6. Compiled python files
    1. Python caches the compiled version of each module in __pycache__ directory under the name module.version.pyc
    2. Python checks the modification date of the source against the compiled version to see if it's out of date and needs to be recompiled.
        1. Above is a completely automatic process
    3. The compiled modules are platform-independent, so the same library can be shared among systems with different architectures.
    4. Python does not check the cache in two circumstances:
        1. Python always recompiles and does not store the result for the module that's loaded directly from the cmd line.
        2. Python does not check the cache if there is no source module.
5. Tips for experts
    1. Use -o or -oo switches on the python cmd to reduce the size of a compiled module
        1. -o switch remove assert statements
        2. -oo switch removes both assert statements and __doc__ strings
        3. You should only use this option if you know what you're doing, because some programs may rely on having these available
    2. A program doesn't run any faster when it is read from a .pyc file than when it is read from a .py file
        1. The only thing that's faster about .pyc files is the speed with which they are loaded.
    3. The module compileall can create .pyc files for all modules in a directory

Standard Modules

7. Standard modules
    1. Some modules are built into the interpreter
        1. For efficiency
        2. Provide access to operating system primitives such as system calls
        3. Is a configuration option depends on the underlying platform
    4. Sys module built into every python interpreter
        1. sys.path is a list of strings that determines the interpreter's search path for modules
        2. sys.path initialized to a default path taken from the environment variable PYTHONPATH
        3. You can modify sys.path by using standard list operations
            1. Import sys
            2. Sys.path.append('/…/..')

Modules

The dir() function

8. dir() function is the built-in function to find out which names a module defines
9. How to list the names of built-in functions and variables

1. Import builtins
2. dir(builtins)

Modules

Packages

10. What is packages?
    1. Packages are a way of structuring python's module namespace by using "dotted module names"
    2. The module name A.B
        1. Submodule named B in a package named A
11. Why packages?
    1. The use of dotted module names saves the authors of multi-module packages like NumPy or Pillow from having to worry about each other's module names
12. How it works?

Modules

Packages

12.How packages works?

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

Modules

Packages

12. How packages works?
    1. When importing the package, python searches through the directories on sys.path looking for the package subdirectory
    2. The __init__.py files are required to make python treate dictionaries containing the file as a packages
        1. __init__.py may be an empty file
        2. __init__.py execute initialization code for package
        3. __init__.py set the __all__ variable
    3. “from package import item”
        1. item can be a submodule or function, class, variable defined in the package
        2. import
            1. Import first tests if the item is defined in the package, if not, it assumes it is a module and attempts to load it. If it fails to find it ImportError exception is raised
        3.
    4.

Modules

Importing * From a Package

13. What happens when the user write ‘from sound.effects import*’?
    1. What we wish to happen?
        1. One would hope that this somehow goes out to the filesystem, finds which submodules are present in the package, and import all
    2. What is the real problem of our wish?
        1. What we wish could take a long time
        2. Importing sub-modules might have unwanted side-effects
    3. The solution, what package author should do?
        1. Package author to provide an explicit index of the package.
        2. What import statement do?
            1. a package’s __init__.py code defines a list named __all__, it is taken to be the list of modules
                1. All the list in __all__ imported when “from package import *” encountered
        3. Package author should keep the list in __all__ up-to-date when a new version of the package is released
            1. __all__ = [‘echo’, ‘surround’, ‘reverse’]
                1. This means “from sound.effects import *” would import he three named submodules of the sound.effects package
        4. What happens if __all__ is not defined?
            1. The “from sound.effects import *” does not import all submodules
            2. It only ensures that the package sound.effects has been imported
4. import * considered bad practice in production code.

Modules

Intra-package references

14. Intra-package reference
15. Packages in multiple directories

Input and Output

Output of a program

1. Output of a program
    1. Printed in a human-readable form
    2. Written to a file for future use
    3. Other possibilities
2. Fancier Output Formatting
    1. 3 ways of writing values:
        1. Expression statement
        2. print() function
        3. write() method of file object
3. Ways to format output
    1. Use formatted string literals, begin a string with f and F before the opening quotation mark or triple quotation mark.
        1. Inside the string, you can write python expression between { and }
    2. str.format() method of strings
        1. use { and } to mark where a variable will be substitued
4. What if you just want a quick display of some variables for debugging purposes?
    1. Using repr() or str() functions can convert any value to a string

Input and Output

Formatted String Literals

5. What is formatted string literals(f-strings for short)?
    1. A string by prefixing the string with f or F and writing expressions as {expression}
6. How to making columns line up?
    1. Passing an integer after the “:” will cause that field to be a minimum number of characters wide
7. Do you know modifiers like ‘!a’ ‘!s’ and ‘!r’?
    1. ‘!a’ applies ascii()
    2. ‘!s’ applies str()
    3. ‘!r’ applies repr()
8. = specifier used to expand an expression to the text of the expression

The String format() method

str.format() method

9. str.format() method
    1. Brackets and characters within them are replaced with the objects passed into the str.format() method.
    2. A number in the brackets used to refer to the position of the object passed into the str.format() method
    3. If keyword arguments are used, their values are referred to by using the name of the argument
    4. Keyword argument and number in bracket combined.
    5. What should do if have a long format string that you don’t want to split up?
        1. Passing the dict and using square brackets [] to access the keys

Input and Output

Manual String Formatting

10. str.ljust() str.rjust() str.center() method
    1. rjust() a string in a field of a given width by padding it with spaces on the left
11. str.zfill() method
    1. Pad a numeric string on the left with zeros
12. Old string formatting
    1. % operator(modulo)
        1. String % values

Reading and Writing Files

open(filename, mode, encoding=None)

13.open(filename, mode, encoding='utf-8')

1. Mode:
    1. A ('r'/'w'/'a'/'r+') string describing the way the file used
    2. 'R' file will only be read
    3. 'W' file for only writing
    4. 'A' opens the file for appending
    5. 'r+' opens file for both reading and writing
2. Platform-specific line endings may be converted may corrupt file
    1. Be very careful to use binary mode when reading and writing
3. Good practice to use the with keyword when open files
    1. If you use with, file is properly closed after its suite finishes
    2. If you not use with, you should call f.close() to close the file
    3. Calling f.write() without using the with keyword or calling f.closse() might result in the arguments of f.write() not being completely written to the disk, even if the program exits successfully.

Input and Output

Methods of File Objects

14. f.read(size)
    1. Read some quantity of data and returns it as a string or bytes.
    2. Size is optional
15. f.readline()
16. Do you want reading line by line from a file?
    1. for line in f:
17. What if you want read all the lines in a list?
    1. list(f)
18. Use f.write() with ‘a’ mode
19. f.tell()
    1. Returns an integer giving the file object’s current position
20. f.seek(offset, whence) to change the file objects’s position

Input and Output

Saving structured data with json

21.How to read/write numbers from a file?

1. Why read() not worked for numbers?
    1. read() method only return strings, have to be passed to a function like int()
        1. What happens if you want to save more complex data types like nested lists and dictionaries?

22.Amazing JSON(javascript object notation)

1. What is called serializing?
    1. json module can take Python data hierarchies, and convert them to string representations,
2. What is call deserializing?
    1. Reconstruting the data from the string representation is called deserializing.
3. What between serializing and deserializing?
    1. The string representing the object may have been stored in a file or data, or sent over a network connection to some distant machine.
4. The JSON format is commonly used by modern applications to allow for data exchange.
    1. programmer used to make it a good choice for interoperability.
5. json.dumps(x)
6. json.dump(x,f)
7. Json.load(f)
8. JSON files must be encoded in UTF-8. Use encoding='utf-8' when opening JSON file as a text file for both reading and writing.

24.Pickle — the pickle module

1. Pickle is a protocol allows the serialization of arbitrarily complex python objects
2. Careful to use it may insecure

Errors and Exceptions

Syntax Errors & exceptions

1. How many kinds of errors?
    1. Mainly two distinguishable kind of errors
        1. Syntax errors
        2. Exceptions
2. What is Syntax Errors?
    1. Known as parsing errors, most common kind
3. What is Exceptions?
    1. Errors detected during execution are called exceptions
        1. ZeroDivisionError
        2. NameError
        3. TypeError
    2. What is traceback(most recent call last) in the error messages?
        1. Traceback is a stack shows the context where the exception occurred
        2. Error messages contains a stack traceback listing source lines

Handling Exceptions

Write programs handle selected exceptions

4. How try works?
    1. The try clause between the try and except keywords is executed.
    2. No exception occurs, except clause is skipped
    3. If exception occurs, if error type match the name after the except keyword, except clause executed.
    4. If exception occurs, if error type not match, it is passed on to outer try statements; if no handler is found, it is an unhandled exception.

Handling Exception

Except in derived class

5. Except clauses reversed or not in classes?
    1. Not reversed
    2. An except clause listing a derived class is not compatible with a base class, see examples pls

Handling exception

BaseException

6. What is BaseException?
    1. All exceptions inherit from BaseException
    2. Use BaseException with extreme caution
        1. BaseException can make a real programming error.

7. When BaseException to be used?
    1. Used to print an error message and then re-raise the exception
        1. Allow the caller to handle the exception

Handle Exceptions

else clause after try except

8. Why need else clause after try except?
    1. It is useful for code that must be executed if the try clause does not raise an exception.
9. Why it is better to add else clause after try except than adding else clause to try clause?
    1. It avoids accidentally catching an exception that wasn’t raised by the code being protected by the try except statement.

Handle Exceptions

Exception’s arguments

10. Exceptions also have arguments?
    1. When exception occurs, it may have an associated value, that is exception’s argument
    2. If an exception has arguments, they are printed as the last part of the message for unhandled exceptions
11. How except clause works?
    1. Except clause may specify a variable after the exception name.
    2. The variable is instance.args that bound to exception instance.
    3. For convenience, the exception instance defines __str__(), so the arguments can be printed directly without having to reference.args
    4. Sometimes, can instantiate an exception first before raising it and add any attributes to it as desired.
    5. Exception handlers handle exceptions both in try clause and inside functions that are called in the try clause.

Handle Exceptions

Raising exceptions

12. What is raise statement used for?
    1. The raise statement allows the programmer to force a specified exception to occur.
    2. The sole argument to raise indicate the exception to be raised.
13. Either an exception instance or exception class
    1. If an exception class is passed, it will be implicitly instantiated by calling its constructor with no arguments
14. What if you need to determine whether an exception was raised but don’t intend to handle it?
    1. A smiple raise statement allows you re-raise the exception

Handle Exception

Exception Chaining

15. What happened if an unhandled exception occurs inside an except section?
    1. It will have the exception being handled attached to it and included in the error message.
16. What is “raise XXX from XXX” used for?
    1. Use from clause in the raise statement is to indicate that an exception is a direct consequence of another.
    2. From clause can be useful when you are transforming exceptions
    3. Allows you disabling automatic exception chaining using the from None idiom
17. Python Standard Library Built-in Exceptions for detail

Handle Exceptions

User-defined Exceptions

18. How to name your own exceptions?
    1. By creating a new exception class
    2. Exceptions named that end in “Error”
19. What exception class can do?
    1. Can do anything any other class can do, but kept simple.
    2. Offering a number of attributes
        1. These attributes used for extracted by handlers for the exception.

Handle Exceptions

Defining Clean-up Actions

20.What finally clause in try statement can do?

1. Intended to define clean-up actions that must be executed under all circumstances
2. The finally clause is useful for releasing external resources(such as files or network connections), regardless of whether the use of the resource was successful

21.How finally clause works?

1. finally clause always runs as the last task before the try statement completes.
2. The finally clause is executed in any event.

22.What happened when an exception occurs?

1. If an exception occurs during execution of the try clause:
    1. The exception may be handled by an except clause
        1. If not handled by except clause:
            1. The exception is re-raised after the finally clause has been executed.
2. If an exception occurs during execution of an except or else clause:
    1. The exception is re-raised after the finally clause has been executed.
3. If the finally clause executes a break, continue, or return statement, exceptions are not re-raised.
4. If the try statement reaches a break, continue or return statement, the finally clause will execute just prior to the break, continue or return statements's execution.
5. If a finally clause include a return statement, the returned value will be the one from the finally clause's return statement, not the value from the try clause's return statement.

Handle Exceptions

Predefined Clean-up Actions

23. Some objects define standard clean-up actions, such as with clause in file operation
24. What happened if open file without with clause and didn't not close() it at the end?
    1. Not an issue in simple scripts, but can be a problem for larger applications
25. How do we know if an object provide predefined clean-up actions?
    1. They will indicate predefined clean-up actions in their documents.

Classes

Means of bundling data and functionality together

1. What happened if you create a new class?
    1. A new class means a new type of object
    2. A new object type allows news instances
    3. A class instance attached attributes to maintaining its state
    4. A class instance have methods to modify its state.
2. Advantages of Classes in python?
    1. Adds classes with some new syntax and semantics
    2. Provide all the standard features of oop:
        1. Class inheritance allows multiple base classes
        2. A derived class can override any methods of its base class,
        3. A derived class can call the methods of a base class with the same name
        4. Class object can contain arbitrary amounts and kinds of data
        5. Dynamic: they are created at runtime, and can be modified further after creation.
3. No shorthands for referencing the objects members from its methods
4. Classes themselves can be objects
5. Built-in types can be used as base classes for extension by the user
6. Most built-in operators with special syntax(arithmetic operators, subscripting) can be redefined for class instances.

Classes

Names and objects

3. What is aliasing?
    1. Aliasing means multiple names can be bound to the same object
4. Benefit of using alias?
    1. alias behave like pointers when involving mutable objects as lists, dictionaries…
        1. Passing an object is cheap since only a pointer is passed.

Classes

Python scopes and namespaces

5. Namespace
    1. Namespace is a mapping from names to objects
    2. Namespace implemented as python dictionaries
    3. Examples of namespace:
        1. Set of built-in names
            1. functions as abs()
            2. Built-in exception names
        2. Global names in a module
4. Important thing to know about namespace is:
    1. There is no relation between names in different namespaces
        1. Two different modules can both define same name function.
5. Namespaces are created at different moments and have different lifetimes
    1. Built-in names created when python interpreter starts up, and is never deleted
    2. Global namesake for a module is created when the module definition is read in; normally, module namespace also last until the interpreter quit.

Classes

Scope

6. Quirks of scopes
    1. Scopes are determined textually:
        1. The actual search for names is done at runtime, the language definition is evolving towards static name resolution at compile time
            1. So, don’t rely on dynamic name resolution
    2. Assignment don’t copy data - they just bind names to objects

Classes

Class objects

7. Class objects
    1. When a class definition is left normally, a class object is created
        1. A class object is a wrapper around the contents of the namespace created by the clan definition
        2. Class object is bound to the class name given in the class definition header
    2. Class objects support two kind of operations:
        1. Attribute references: obj.name
        2. Instanitaion
            1. Class instantiation uses function notation.
                1. Think that the class object is a parameterless function that returns a new instance of the class
                2. X = MyClass()
                    1. Create a new instance of the class and assigns this object to the local variable x
            2. Instantiation operation creates an empty object
3. __init__() method
    1. Class instantiation automatically invokes __init__() for the newly created class instance.
    2. __init__() method may have arguments for greater flexibility

Classes

Method Objects

8. Method objects
    1. Method can be stored away and called at a later time
9. What happened when a method is called?
    1. The instance object is passed as the first argument of the function
    2. Calling a method with a list of n arguments is equivalent to calling the corresponding function with an argument list that is created by inserting the method's instance object before the first argument.
        1. x.f() == MyClass.f(x)
3. A method object is created by pointers to the instance object and the function object just found together in an abstract object
4. When a method object is called with an argument list, a new argument list is constructed from the instance object and the argument list, the function object is called with this new argument list.

Classes

Class and Instance Variables

10. Instance variables
    1. Instance variables are for data unique to each instance
11. Class variables
    1. Class variables are for attributes and methods shared by all instances of the class
    2. Side effects of shared class variables
        1. When class variables is list or dictionaries, tricky may happen, see examples in .py file

Classes

Random Remarks

12. Random Remarks

    1. What happened if the same attribute name occurs in both an instance and in a class?
        1. Attribute lookup prioritizes the instance
    2. Assigning a function object to a local variable in the class is ok
    3. Methods may call other methods by using method attributes of the self argument.
    4. Each value is an object, has a class(type).
        1. It is stored as object._class_

Classes

Inheritance

13. Syntax of a derived class `class DerivedClassName(BaseClassName):`
    1. BaseclassName must be defined in a scope containing the derived class definition
    2. The base class name allows arbitrary expressions `class DerivedClassName(modname.BaseClassName):`
    3. When the class object is constructed, the base class is remembered.
14. How to find attribute references?
    1. If a requested attribute is not found in the class, the search proceeds to look in the base class, and applied recursively if the base class itself is derived from some other class
15. How to resolve method references?
    1. The class attributes is searched descending down the chain of base classes, method reference is valid if this yields a function object.
    2. Derived classes may override methods of their base classes
        1. Methods have no special privilege when calling other methods of the same object
            1. A method of a base class that calls another method defined in the same base class may end up calling a method of a derived class that overrides it.
        2. A overriding method in a derived class may to extend rather than simply replace the base class method at the same time.
    3. How to call the base class method directly?
        1. Just call: BaseClassName.methodname(self, arguments)
            1. This only works if the base class is accessible

Classes

Two built-in functions works with inheritance

16. Two built-in functions works with inheritance
    1. isinstance() to check an instance's type
        1. isinstance(obj, int) will be true only if obj.__class__ is int or some class derived from int
    2. issubclass() to check class inheritance:
        1. issubclass(bool, int) is True, since bool is a subclass of int.
        2. issubclass(float, int) is False, since float is not subclass of int.

Classes

Multiple inheritance

17.Multiple inheritance

1. class DerivedClassName(Base1, Base2, Base3)
2. How to search for attributes inherited from parent class?
    1. Depth-first, left-to-right
3. super() approach?
    1. The method resolution order changes dynamically to support cooperative calls to super()
4. Dynamic ordering when diamond relationships between multiple inheritance.
5. How to keep the base class from being accessed more than once in diamond relationship?
    1. The dynamic algorithm linearizes the search order and preserves the left-to-right ordering specified in each class.
    2. Monotonic is that a class can be subclassed without affecting the precedence order of its parents

Classes

Private Variables

18. Private Variables
    1. A name prefixed with an underscore(_spam) should be treated as a non-public part of the API
19. What is name mangling?
    1. Any identifier of the form __spam(two underscores) is textually replaced with _classname__spam
    2. Mangling is done without regard to the syntactic position of the identifier.
    3. Name mangling is helpful for letting subclasses override methods without breaking intraclass method calls.
    4. Mangling rules are designed to avoid accidents
20. exec() / eval()/ global statement/ getattr()/ setattr()/ delattr()/ __dict__
    1. The code passed above functions does not consider the classname of the invoking class to be the current class.
    2. Is restricted to code that is byte-compiled together.

Classes

Odds and Ends

21. How python implement data type similar to C struct?

```python
#python's data type similar to C 'struct', bunding together a few named data items
#An empty class definition
class Employee:
    pass

john = Employee()    #Create an empty employee record

#Fill the fields of the record
john.name = 'John Doe'
john.dept = 'Computer lab'
john.salary = 2000

print(john.name)
```

Classes

iterators

22. The use of iterators pervades and unifies python
23. How for statement works?
    1. The for statement calls iter() on the container object
    2. iter() function returns an iterator object that defines the method __next__() which accesses elements in the container one at a time.
    3. When no more elements, __next__() raises a StopIteration exception which tells the for loop to terminate
        1. You can call the __next__() method by using the next() built-in function

Classes

How to add iterator behavior to your classes

24.How to add iterator behavior to your classes?

1. Define an __iter__() method which return an object with a __next__() method
2. If the class defines __next__(), then __ite__() can just return self

Classes

Generators

25.What is Generators?

1. Generators are a simple and powerful tool for creating iterators.
2. Generators are written like regular functions but use yield statement whenever they want to return data.
3. Each time next() is called on it, the generator resumes where it left off
4. The generator remembers all the data values and which statement was last executed
5. The __iter__() and __next__() methods are created automatically in Generator.
6. Local variables and execution state are automatically saved between calls.
7. When generators terminate, they automatically raise StopIteration

Classes

Generator Expressions

26. Use Generator Expressions

    1. Simple generators can be coded as expressions using a syntax similar to list comprehensions but with parentheses instead of square brackets.

Brief Tour of the Standard Library

Tour of the Standard Library - part 1

1. What's mentioned in Brief Tour of STL?
    1. Operating System Interface
    2. File wildcards
    3. Command Line Arguments
    4. Error Output Redirection and Program Termination
    5. String Pattern Matching
    6. Mathematics
    7. Internet Access
    8. Dates and Times
    9. Data Compression
    10. Performance measurement
    11. Quality Control
    12. Batteries included

Brief tour of the standard library

Operating System Interface

2. What we need to know about file/dir operations?
    1. os module
        1. Why Use import os style instead of from os import *
            1. Import os will keep os.open() from shadowing the built-in open() function which operates much differently
    2. Built-in functions dir() and help() are useful when working with large modules like os
    3. Use shutil module for daily file and directory management task

Brief Tour of the standard library

File wildcards

3. Glob module
    1. Glob module provides a function for making file lists from directory wildcard searches

Brief Tour of the Standard Library

Command Line Arguments

4. print(sys.argv) : Cmd line arguments stored in sys modules's argv attribute as a list.
5. argparse module provide a more sophisticated mechanism to process command line arguments.

Brief tour of standard library

Error Output Redirection and Program Termination

6. Using sys.stderr.write(“”) to emitting warnings and error messages
7. How to terminate a script?

    Sys.exit()

Brief tour of the standard library

String Pattern Matching

8. re module provides regular expression tools for advanced string processing
9. Simply use str.replace() when simple capabilities are needed.

Brief Tour of Standard Library

Mathematics

10. Use math module to handle floating point
11. Use random module to making random selections
12. Use statistics module to calculates basic statistical properties
13. Use modules in Scipy project for numerical computations

Brief Tour of Standard Library

Internet Access

14. Use urllib.request module to retrieving data from URLs
15. Use smtplib module to sending mail

Brief Tour of Standard Library

Dates and Times

16. Use datetime module for manipulating and formatting dates and times

Brief Tour of Standard Library

Data Compression

17. Using modules: glib, gzip, bz2, Lama, zipfile and tarfile to data achiving and data compression.

Brief Tour of Standard Library

Performance Measurement

18. Use timeit module demonstrates a modest performance advantage, timeit module to measure time granularity.
19. Use the tuple packing and unpacking feature.
20. Use profile and pstats modules to identify time critical sections in larger blocks of code

Brief Tour of Standard Library

Batteries Included

21. What batteries included used for?
    1. Batteries included is best seen through the sophisticated and robust capabilities of its larger package.
    2. Examples:
        1. xmlrpc.client and xmlrpc.server modules
            1. Silly names, no handling of xml is needed
        2. Email package for managing email messages
            1. Has a complete toolset for building or decoding complex message structures
        3. Json package: robust support for parsing data interchange format
        4. Cvs module supports direct reading and writing of files in Comma-Separated value format
        5. Xml package
        6. Sqlite3 module for SQLite database library
        7. Gettext, locale, codec packages for internationalization

Brief Tour of Standard Library

Quality Control

21. Write test for each function is one approach for developing high quality software.
22. Use doctest module to scanning a module and validating tests embedded in a program's docstrings.
23. Use unites module for more comprehensive set of tests
    1. Calling from the command line invokes all tests.
        1. warning, log file not found starting a new oneLaunching unittests with arguments python -m unittest /Users/lina/code/core-python/Python_Tutorial_10_stdlib.py in /Users/lina/code/core-python

Brief Tour of The Standard Library

Advance modules - Output Formatting

1. Use the reprlib module for abbreviated displays of large or deeply nested containers.
    1. reprlib.repr() method returns the object representation in string format.
2. The pprint module known as “pretty printer” that can add line breaks and more clearly reveal data structure.
3. textwrap module formats paragraphs of text to fit a given screen width.

Brief Tour of the Standard Library

Advanced module- Templating

4. Users can substitute placeholder in Template class in string module
    1. Users can customize their applications without having to alter the application
    2. Syntax the placeholders like ${variablename}
    3. Use substitute()(in Template class) method properly
        1. substitute() method may raise KeyError, if a placeholder is not supplied in a dictionary or a keyword argument.
    4. safe_substitute() method for mail-merge style applications
        1. It will leave placeholders unchanged if data is missing.
    5. Template subclasses can specify a custom delimiter
        1. Use percent signs for placeholders like current date, image sequence number, or file format.
    6. Template can separating program logic from the details of multiple output formats.
        1. Can substitute custom templates for XML files, plain text reports, and HTML web reports.

Brief Tour of Standard Library

Working with Binary Data Record Layouts

5. Use pack() and unpack() method in struct module for working with variable length binary record formats.
6. How to loop through header information in zip file without using the zipfile module

1. See example in .py file

Brief tour of standard library

Multi-threading

7. Threading module can run tasks in background while the main program continues to run
8. What's the challenge of multi-threaded applications?
    1. Coordinating threads that share data or other resources.
        1. The threading module provides a number of synchronization primitives including locks, events, condition variables, and semaphores.
    2. Multithreading related modules are powerful, minor designs errors can result in problems that are difficult to reproduce.
        1. How to resolve this? How to coordinate tasks?
            1. Concentrate all access to a resource in a single thread and then use the queue module to feed that thread with requests from other threads.
            2. Applications using queue objects for inter-thread communication and coordination are easier to design, more readable and more reliable.

Brief Tour of standard library

Logging

9. Use logging module to manage log messages to different files
    1. Informational and debugging messages are suppressed
    2. The output is sent to standard error.
    3. Output options including routing messages through email, datagrams, sockets, or to an HTTP server.
    4. Message priority:
        1. DEBUG
        2. INFO
        3. WARNING
        4. ERROR
        5. CRITICAL

Brief tour of standard library

Weak references

10. Python does automatic memory management, the memory if freed shortly after the last reference to it has been eliminated.
11. How to track objects only as long as they are being used by something else?
    1. Problem is just tracking them creates a reference that makes them permanent
    1. How to solve this problem?
        1. weakef module can tracking objects without creating a reference.
        2. Object is automatically removed from weakef table when the object is no longer needed.
        3. Catching objects that are expensive to create.

Brief Tour of Standard Library

Tools for working with lists

12. Python provide built-in list type, but sometimes need for alternative implementations with different performance trade-offs.
13. array.array() object like a list, but stores only homogeneous data and stores it ore compactly.
14. collections.deque() object like a list, but with faster appends and pops from the left side but slower lookups in the middle.
    1. collections.deque are well suited for implementing queues and breadth first tree searches.
15. bisect module for manipulating sorted lists
16. heapq module for implementing heaps based on regular lists, the lowest valued entry is always kept at position zero.

Brief Tour of Standard Library

Decimal floating point arithmetic

17. Use decimal module for decimal floating point arithmetic.
18. decimal module is really used for:
    1. Financial applications require exact decimal representation
    2. Control over precision
    3. Legal or regulatory requirements
    4. Tracking of significant decimal places
    5. Applications where the user expect the results to match calculations done by hand.
    6. Decimal module provides arithmetic with as much precision as needed

Virtual Environments and Packages

Why python need virtual environment?

1. Why python need virtual environment?
    1. Some python application need a specific version of a library.
    2. Python you currently installation may not meet the requirement of every application.
        1. The resolution is to create a virtual environment
2. What is virtual environment?
    1. A virtual environment is a self-contained directory tree that contains a python installation for a particular version of python, plus a number of additional packages.
    2. Different applications can use different virtual environments.
3. How to create virtual environment?
    1. Use the venv module
    2. Venv will install the most recent version of Python that you have available.
    3. To create a virtual environment:
        1. Decide upon a directory where you
        2. want to place it.
        3. Run the venv module as a script with the directory path: python3 -m venv tutorial-env
        4. python3 -m venv tutorial-env will create. The tutorial-env directory, and also create directories inside it containing a copy of the Python interpreter and various supporting files.
        5. .venv is typically hidden in your shell
        6. Once you've created a virtual environment, you may activate it by running: source tutorial-env/bin/activate
        7. Activating the virtual environment will change your shells prompt to show what virtual environment you're using, and modify the environment so that running python will get you that particular version and installation of Python.

Virtual Environment and packages

Managing packages with pip

4. How can we install, upgrade, and remove packages using program called pip?
    1. Install the latest version of a package by specifying a package's name:
        1. python -m pip install novas
    2. Install a specific version by:
        1. Python -m pip install requests==2.6.0
    3. Update the package:
        1. Python -m pip install —upgrade requests
    4. Uninstall package
        1. pip uninstall
    5. Display information about a particular package
        1. pip show
    6. Display all of the packages installed in the virtual environment
        1. pip list
    7. Pip freeze will produce a similar list of the installed packages, but the output uses the format that pip install expects
        1. Pip freeze > requirements.txt
    8. User can install all the necessary packages with install -r
        1. Python -m pip install -r requirements.txt