English is my ESL, and this is the first time i write on this websites,`````So , you know ````````.

How to use google or doc of python to get answer.

【stackoverflow】search through stackoverflow, after get the answer, there might be some reference towards doc of python and we can see the detail.s

OR search the doc, judge which one is correct.

Sometimes it seems

# corrigendum勘误表? 谁错？？

|  |  |
| --- | --- |
|  |  |
| list.pop([]) | list.pop() |
|  |  |

# Spyder:

Some tricks for spyder using

Reset spyder: Tools>Reset spyder to factory defaults; anaconda prompt: I forgot it, wait for

Set current working directory: Tools>Preferences>Current working directory.

# Question:

1. is that something about argument and parameter?

a=1

print(a)

print(a=2)

>>>1

>>> TypeError: 'a' is an invalid keyword aqrgument for this function

1. similarly,

type(a=1)

>>> TypeError: type() takes 1 or 3 arguments

1. what does it mean? for语句好像是用multiplication来创建h行[None]\*w

w, h = 2, 3

A = [[**None**] \* w **for** i **in** range(h)]

>>> [[None, None], [None, None], [None, None]]

1. ?? Concatenating immutable sequences always results in a new object. This means that building up a sequence by repeated concatenation will have a quadratic runtime cost in the total sequence length. To get a linear runtime cost, you must switch to one of the alternatives below:
2. why tuple can be concatenated though it is immutable
3. handout8 :???if (b%2)\*2==b
4. ??? **Q:** the complexity of searching a sorted list: time of indexing of a linked list is linear while the indexing of the list of which the pointer of every element is successive stored in memory.**A: linked list searching cannot through leaping, one by one**
5. **Generator:** in the description in page yield expressions, what is it about the concrete implementation of the generator works, like how it controls the execution of the generator function, e.g. some words suspended, resume etc.
   1. [4.5. Iterator Types](file:///D:\003%E6%9C%BA%E5%99%A8%E4%BA%BA%E5%B7%A5%E7%A8%8B%E5%B8%88%E5%AD%A6%E4%B9%A0%E8%AE%A1%E5%88%92\%E8%AF%BE%E7%A8%8B\%E8%AE%A1%E7%AE%97%E6%9C%BA\%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6%E5%8F%8A%E7%BC%96%E7%A8%8B%E5%AF%BC%E8%AE%BA\python-3.6.5-docs-html\library\stdtypes.html#iterator-types)>>>[generator](file:///D:\003%E6%9C%BA%E5%99%A8%E4%BA%BA%E5%B7%A5%E7%A8%8B%E5%B8%88%E5%AD%A6%E4%B9%A0%E8%AE%A1%E5%88%92\%E8%AF%BE%E7%A8%8B\%E8%AE%A1%E7%AE%97%E6%9C%BA\%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6%E5%8F%8A%E7%BC%96%E7%A8%8B%E5%AF%BC%E8%AE%BA\python-3.6.5-docs-html\glossary.html#term-generator)>>>[generator iterator](file:///D:\003%E6%9C%BA%E5%99%A8%E4%BA%BA%E5%B7%A5%E7%A8%8B%E5%B8%88%E5%AD%A6%E4%B9%A0%E8%AE%A1%E5%88%92\%E8%AF%BE%E7%A8%8B\%E8%AE%A1%E7%AE%97%E6%9C%BA\%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6%E5%8F%8A%E7%BC%96%E7%A8%8B%E5%AF%BC%E8%AE%BA\python-3.6.5-docs-html\glossary.html#term-generator-iterator)>>> [yield](file:///D:\003%E6%9C%BA%E5%99%A8%E4%BA%BA%E5%B7%A5%E7%A8%8B%E5%B8%88%E5%AD%A6%E4%B9%A0%E8%AE%A1%E5%88%92\%E8%AF%BE%E7%A8%8B\%E8%AE%A1%E7%AE%97%E6%9C%BA\%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6%E5%8F%8A%E7%BC%96%E7%A8%8B%E5%AF%BC%E8%AE%BA\python-3.6.5-docs-html\reference\simple_stmts.html#yield) >>>[Yield expressions](file:///D:\003%E6%9C%BA%E5%99%A8%E4%BA%BA%E5%B7%A5%E7%A8%8B%E5%B8%88%E5%AD%A6%E4%B9%A0%E8%AE%A1%E5%88%92\%E8%AF%BE%E7%A8%8B\%E8%AE%A1%E7%AE%97%E6%9C%BA\%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%A7%91%E5%AD%A6%E5%8F%8A%E7%BC%96%E7%A8%8B%E5%AF%BC%E8%AE%BA\python-3.6.5-docs-html\reference\expressions.html#yieldexpr)
6. Class: After\_\_cmp\_\_ was defined in class, why can python directly execute code: q<p; Similarly,, after \_\_str\_\_ was defined, print (object) can directly invoke the method \_\_str\_\_, is it sth. about double underscore
7. global variable and local variable, mutating the variable while iterating its elememnts???

Some properties of python, especially distinguished with other languages, like C or others, like matlab (depends on what i've learnt TT)]] it seems that it is  object-oriented language as well as general language.

From  <http://www.techbeamers.com/python-tutorial-step-by-step#list-of-python-features>

# 0About OOP, oriented-object programming

Code Quality. //  ````// Scientific And Numeric Computing. //  No Explicit Declaration.

State Of The Art OOP Support.

It Seems That It Is  Object-Oriented Language As Well As General Language.

## OOP(From thirty essential questions about python):

 It lays down a model which derive solution by creating objects, defining relationships, and binding data. it is a bottom-up problem-solving technique which seeks a blueprint of the solution at the onset and leaves implementation for the later.

the procedural programming methodology: it takes on a top-down approach and solves one problem at a time while splitting it into smaller ones. On the other hand

## about OOP(From Documentation » The Python Language Reference » 3. Data model)

Objects are Python’s abstraction for data. All data in a Python program is represented by objects or by relations between objects. (In a sense, and in conformance to Von Neumann’s model of a “stored program computer,” code is also represented by objects.)

Every object has an identity, a type and a value. An object’s identity never changes once it has been created; you may think of it as the object’s address in memory. The ‘is’ operator compares the identity of two objects; the id() function returns an integer representing its identity.

# 1 Some common statements for validation

id()returns the memory address of the variable

type() returns the data type of the variable

# 2 Python Keywords, Identifiers And Variables – Fundamentals

Keywords

Identifiers

Variables: variables don’t require declaration but initialization before use. And some details.

some questions:

{

An object is just a region of memory which can hold the following.

* A type designator to reflect the object type.1
* The reference counter which determines when it’s OK to reclaim the object.

}

# 3 Understand Python Statement, Expression & Indentation

The properties of Expression resemble C, but one point that expression can also have a call to a function which evaluates results.

eg.

# Using eval in an expression

>>> eval( "2.5+2.5" )

5.0

for assignment statements,

python will allocate the same memory address for variables in two scenarios:

the strings don’t have whitespaces and contain less than 20 chars;

in case of integers ranging between -5 to +255

eg.>>>test1=learn\... test2=learn\...the results would be the addresses of them are the same.

Implicit Line Continuation & Explicit Line Continuation and

the syntax about tuple: () for number array, []for single char array; 两个元组相加，是扩展

indentation: four spaces replacing {} to mark a block of code

multiline statements

explicit:末尾加\

implicit:用{},[],()，这个不太清楚，建议：查手册

# 4 How To Write Comment And Multiline Comment In Python

Primary point:

indentation is very important for docstring(document string)

first, the usage of the pound/hash/octothorpe symbol # is the same as other symbol like // in any other language.

second, what python has different from others is docstring

1, The strings beginning and ending with triple quotes

2, they are still regular strings except the fact that they could spread to multiple lines, i.e. they are executable statements. And they can be accessed using : myobj.\_\_doc\_\_

2, docstring is placed immediately after a function or class definition or on top of a module (when I place the docstring after the first statements of a function, the result of statement”print(theFunction.\_\_doc\_\_)” is none)

# 5 Python Data Types – Learn From Basic To Advanced

dynamic typing: determine the data type of a literal directly from the syntax at runtime

Everything including variables, functions, modules in Python is **an object**. Another interesting fact is that variables don’t have types instead they are just labels in Python. It is the value which gets associated with a type. Hence, the same variable, the label can refer values of different Python data types.

Numbers:

complex; type(), isinstance(), complex()

strings:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| lists | ordered sequence, enable indexing and repetition; heterogeneous collection of items of vared data types; allow nesting; slicing a list | **mutable** |
| tuples |  | more lightweight than lists;  python uses tuples to return multiple values from a function.  as a single container to stuff multiple things;  as a key in ditionary  **immutable;** |
| set |  | It implements a highly optimized method that checks whether the container hosts a specific element or not.  {}  supports mathematical operations like union, intersection, and symmetric difference;  unordered collection;  don’t allow repetition |
| dictionaries |  | keys map to values |

## Common Sequence Operations

??? Concatenating immutable sequences always results in a new object. This means that building up a sequence by repeated concatenation will have a quadratic runtime cost in the total sequence length. To get a linear runtime cost, you must switch to one of the alternatives below:

* if concatenating [str](https://docs.python.org/3/library/stdtypes.html#str) objects, you can build a list and use [str.join()](https://docs.python.org/3/library/stdtypes.html#str.join) at the end or else write to an [io.StringIO](https://docs.python.org/3/library/io.html#io.StringIO) instance and retrieve its value when complete
* if concatenating [bytes](https://docs.python.org/3/library/stdtypes.html#bytes) objects, you can similarly use [bytes.join()](https://docs.python.org/3/library/stdtypes.html#bytes.join) or [io.BytesIO](https://docs.python.org/3/library/io.html#io.BytesIO), or you can do in-place concatenation with a [bytearray](https://docs.python.org/3/library/stdtypes.html#bytearray) object.[bytearray](https://docs.python.org/3/library/stdtypes.html#bytearray) objects are mutable and have an efficient overallocation mechanism
* if concatenating [tuple](https://docs.python.org/3/library/stdtypes.html#tuple) objects, extend a [list](https://docs.python.org/3/library/stdtypes.html#list) instead
* for other types, investigate the relevant class documentation

For the concatenation of string, we can get the results below using str.join()

>>>string='what does it mean?

>>>string.join('111')

>>>Out[20]: '1what does it mean?1what does it mean?1'

??? index raises [ValueError](https://docs.python.org/3/library/exceptions.html#ValueError) when x is not found in s. Not all implementations support passing the additional arguments i and j. These arguments allow efficient searching of subsections of the sequence. Passing the extra arguments is roughly equivalent to using s[i:j].index(x), only without copying any data and with the returned index being relative to the start of the sequence rather than the start of the slice.

??? 

ll.pop([2])

TypeError: 'list' object cannot be interpreted as an integer

hash table?

|  |
| --- |
| what is hash table? |
| hash table?  ???Creating a set, call the built-in set() function with a sequence or any iterable object.  >>> sample\_set = set("Python data types") >>> type(sample\_set) <class 'set'> >>> sample\_set {'e', 'y', 't', 'o', ' ', 'd', 's', 'P', 'p', 'n', 'h', 'a'} |

# 6 Best Of Python Strings, Functions And Examples

Some question

for (iterating) using for we can iterate through all the characters of the string

var1 = 'Python'

for var in var1:

print (var)

# P

# y

# t

# h

# o

# n

笔记1

关于

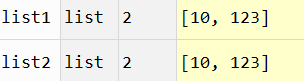
# 30 Essential Python Interview Questions You Should Know before quiz

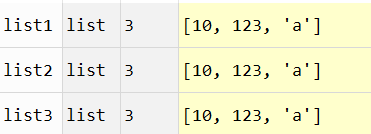
1. **Q:** the code is shown below, and the results follow it. Why would it happen?

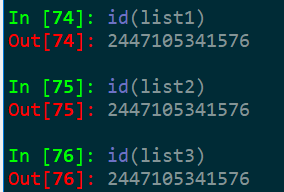
**Test:** if the line “list2=extendList(123)”, and we separately run the code: 1st, run the definition of the function; 2nd, run the second cell line by line, we could get the results as the following in the variable explorer ; 3rd find the ID of each list, id(list1) id(list2) id(list3). Then, the results are shown below.

**Besides,** >>>lists=[[]]\*3 >>>lists[[],[],[]] >>>lists[0].append(3)>>>lists [[3],[3],[3]] >>>id(lists[0]) >>>id(lists(1)) >>>id(lists[2]) all of these will get the same address, because which yields an empty sequence of the same type as s). Note that items in the sequence s are not copied; they are referenced multiple times. Even for >>>lists=[[3]]\*3 >>>id(lists[1]) >>>id(lists[2])









**Original Answer:** However, the flow is like that a new list gets created once after the function is defined. And the same list is used whenever someone calls the extendList method without a list argument. It works like this because the calculation of expressions (in default arguments) occurs at the time of function definition, not during its invocation.

1. \*\*\*\*\***source code**\*\*\*\*\*
2. def extendList(val, list=[]):
3. list.append(val)
4. return list
5. #%%
6. list1 = extendList(10)
7. list2 = extendList(123,[])
8. list3 = extendList('a')
9. #%%
10. print(list1 = %s", list1)
11. print (list2 = %s",list2)
12. print (list3 = %s" ,list3)
13. \*\*\*\*\***results**\*\*\*\*\*
14. list1 = [10, 'a']
15. list2 = [123]
16. list3 = [10, 'a']

# Quiz1 after learning data type (using time)

## top ten questions

|  |
| --- |
| my answer: A##CA DBCDB  correct: CABCC DCCDA |

for the answer or the reason why I was wrong, I’ve been browsing a lot of websites, including the official website (non-official), like The Python Standard Library in python’s documentation, the tutorial I’ve been following,

what should the best reference website for me, digging, keep digging

what strategy should I take?

# Every-moment experiment

In a function, every final result should represent a response to coder. i.e. every path corresponding possibilities should contain stamen “return”, so we can test the value of the function to see which consequence the function perform.

for several type of sequence:

{k:d[k] for k in set(d).intersection(l)}

In [22]: %%timeit

l = xrange(100000)

{k:d[k] for k in l}

....:

100 loops, best of 3: 11.5 ms per loop

%reset –f: clear all variables

common testing tricks and code, including example:

【in】and 【not in】arecate

???【is】和【is not】compare whether two objects are really the same object; this only matters for mutable objects like lists.

‘C’<’c’>>>True; 1<2 >>>True

id() test the address

type(variable)

for string

for datatype:

isinstance()

len()

check the variable explorer

check the namespace

# 二tutorial in python.org

## In For Loop

意思是使用words[:]时，interpreter 会首先创建一个副本，接着迭代副本，然后再插入给words，用words迭代再插入自己的话就是每插入一个又多一个，永远迭代不完。 If you need to modify the sequence you are iterating over while inside the loop (for example to duplicate selected items), it is recommended that you first make a copy. Iterating over a sequence does not implicitly make a copy. The slice notation makes this especially convenient:

>>>

**>>> for** w **in** words[:]: *# Loop over a slice copy of the entire list.*

**...**  **if** len(w) > 6:

**...**  words.insert(0, w)

**...**

**>>>** words

['defenestrate', 'cat', 'window', 'defenestrate']

With for w in words:, the example would attempt to create an infinite list, inserting defenestrate over and over again.

## Defining Function

**def** parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):

print("-- This parrot wouldn't", action, end=' ')

print("if you put", voltage, "volts through it.")

print("-- Lovely plumage, the", type)

print("-- It's", state, "!")

special invalid example:

parrot(voltage=5.0, 'dead') *# non-keyword argument after a keyword argument*

*# i.e. In a function call, keyword arguments must follow positional arguments.*

Some arguments might be passed to the function with several parameters as a tuple or dictionary

**def** cheeseshop(kind, \*arguments, \*\*keywords):

print("-- Do you have any", kind, "?")

print("-- I'm sorry, we're all out of", kind)

**for** arg **in** arguments:

print(arg)

print("-" \* 40)

**for** kw **in** keywords:

print(kw, ":", keywords[kw])

cheeseshop("Limburger", "It's very runny, sir.",

"It's really very, VERY runny, sir.",

shopkeeper="Michael Palin",

client="John Cleese",

sketch="Cheese Shop Sketch")

1,\*name tuple

2, \*\*name dictionary

3, \*\*name must be after \*name ?

### ???unpacking argument lists

The reverse situation occurs when the arguments are already in a list or tuple but need to be unpacked for a function call requiring separate positional arguments. For instance, the built-in [range()](https://docs.python.org/3.8/library/stdtypes.html#range) function expects separate start and stop arguments. If they are not available separately, write the function call with the \*-operator to unpack the arguments out of a list or tuple:

**>>>** list(range(3, 6)) *# normal call with separate arguments*

[3, 4, 5]

**>>>** args = [3, 6]

**>>>** list(range(\*args)) *# call with arguments unpacked from a list*

[3, 4, 5]

???

The above example uses a lambda expression to return a function. Another use is to pass a small function as an argument:

pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]

pairs.sort(key=**lambda** pair: pair[1])

pairs

sort(\*, key=None, reverse=False)

This method sorts the list in place, using only < comparisons between items. Exceptions are not suppressed - if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

sort() accepts two arguments that can only be passed by keyword (keyword-only arguments):

key specifies a function of one argument that is used to extract a comparison key from each list element (for example, key=str.lower). The key corresponding to each item in the list is calculated once and then used for the entire sorting process. The default value of None means that list items are sorted directly without calculating a separate key value.

### ???multi-line docstring

The Python parser does not strip indentation from multi-line string literals in Python, so tools that process documentation have to strip indentation if desired. This is done using the following convention. The first non-blank line after the first line of the string determines the amount of indentation for the entire documentation string. (We can’t use the first line since it is generally adjacent to the string’s opening quotes so its indentation is not apparent in the string literal.) Whitespace “equivalent” to this indentation is then stripped from the start of all lines of the string. Lines that are indented less should not occur, but if they occur all their leading whitespace should be stripped. Equivalence of whitespace should be tested after expansion of tabs (to 8 spaces, normally).

5, Data Structures

5.1.4 Nested List comprehensions

The following list comprehension will transpose rows and columns:

Actually, I guess that the order of execution is ‘I in range(4)’ first, “row in matrix next”, or say so, nest list comprehension after the mother list comprehension.

Not scanning and execute the code from left to right.

**>>>** [[row[i] **for** row **in** matrix] **for** i **in** range(4)]

[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]

5,3 Tuples and Sequences

???The statement t = 12345, 54321, 'hello!' is an example of tuple packing: the values 12345, 54321 and 'hello!' are packed together in a tuple. The reverse operation is also possible:

**>>>** x, y, z = t

This is called, appropriately enough, sequence unpacking and works for any sequence on the right-hand side. Sequence unpacking requires that there are as many variables on the left side of the equals sign as there are elements in the sequence. Note that multiple assignment is really just a combination of tuple packing and sequence unpacking.

dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys. Tuples can be used as keys if they contain only strings, numbers, or tuples; if a tupl e contains any mutable object either directly or indirectly, it cannot be used as a key. You can’t use lists as keys, since lists can be modified in place using index assignments, slice assignments, or methods like append() and extend().

It is sometimes tempting to change a list while you are looping over it; however, it is often simpler and safer to create a new list instead.

Note that in Python, unlike C, assignment cannot occur inside expressions. C programmers may grumble about this, but it avoids a common class of problems encountered in C programs: typing = in an expression when == was intended.

### 5.8. Comparing Sequences and Other Types

As the code of the first line shown following, We can see that If all items of two sequences compare equal, the sequences are considered equal. If one sequence is an initial sub-sequence of the other, the shorter sequence is the smaller (lesser) one.

(1, 2) < (1, 2, -1)

(1, 2, ('aa', 'ab')) < (1, 2, ('abc', 'a'), 4)

# 6,Modules

???On the other hand, if you know what you are doing you can touch a module’s global variables with the same notation used to refer to its functions, modname.itemname.

**Note：** For efficiency reasons, each module is only imported once per interpreter session. Therefore, if you change your modules, you must restart the interpreter – or, if it’s just one module you want to test interactively, use importlib.reload(), e.g. import importlib; **importlib.reload(modulename).**

???how to use “Executing modules as scripts”

**if** \_\_name\_\_ == "\_\_main\_\_":

**import** **sys**

fib(int(sys.argv[1]))

python fibo.py 50

???以后再看

6.1.3. “Compiled” Python files

To speed up loading modules, Python caches the compiled version of each module in the \_\_pycache\_\_ directory under the name module.version.pyc, where the version encodes the format of the compiled file; it generally contains the Python version number. For example, in CPython release 3.3 the compiled version of spam.py would be cached as \_\_pycache\_\_/spam.cpython-33.pyc. This naming convention allows compiled modules from different releases and different versions of Python to coexist.

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. This is a completely automatic process. Also, the compiled modules are platform-independent, so the same library can be shared among systems with different architectures.

Python does not check the cache in two circumstances. First, it always recompiles and does not store the result for the module that’s loaded directly from the command line. Second, it does not check the cache if there is no source module. To support a non-source (compiled only) distribution, the compiled module must be in the source directory, and there must not be a source module.

Some tips for experts:

You can use the -O or -OO switches on the Python command to reduce the size of a compiled module. The -O switch removes assert statements, the -OO switch removes both assert statements and \_\_doc\_\_ strings. Since some programs may rely on having these available, you should only use this option if you know what you’re doing. “Optimized” modules have an opt- tag and are usually smaller. Future releases may change the effects of optimization.

A program doesn’t run any faster when it is read from a .pyc file than when it is read from a .py file; the only thing that’s faster about .pyc files is the speed with which they are loaded.

The module compileall can create .pyc files for all modules in a directory.

There is more detail on this process, including a flow chart of the decisions, in PEP 3147.

6.2 Standard Modules

The variable sys.path is a list of strings that determines the interpreter’s search path for modules. It is initialized to a default path taken from the environment variable PYTHONPATH, or from a built-in default if PYTHONPATH is not set.

we can modify the sys.path using standard list operations:

**import** **sys**

sys.path.append('/ufs/guido/lib/python')

6.4 ???（不完全懂）packages

Note that when using from package import item, the item can be either a submodule (or subpackage) of the package, or some other name defined in the package, like a function, class or variable.

Contrarily, when using syntax like import item.subitem.subsubitem, each item except for the last must be a package

### how to organize packages

???(Including Importing)Although certain modules are designed to export only names that follow certain patterns when you use import \*, it is still considered bad practice in production code. From:6.4.1 Importing \* From a Package

# 7. Input and Output

**Test:**

repr(x, y, ('spam', 'eggs'))

>>> repr() takes exactly one argument (3 given)

repr((x, y, ('spam', 'eggs')))

>>> "(32.5, 40000, ('spam', 'eggs'))"

because The argument to repr() may be any Python object, including tuple

**And** **also**

**>>>** *# The repr() of a string adds string quotes and backslashes:*

**...** hello = 'hello, world**\n**'

**>>>** hellos = repr(hello)

**>>>** print(hellos)

'hello, world\n'

For str.format(), there are lots of properties which would not listed here one by one but something strange, at least I think it strange.

**Test**

table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}

**for** name, phone **in** table.items():

print('*{0:10}* ==> *{1:10d}*'.format(name, phone))

**Result**

Jack ==> 4098

Dcab ==> 7678

Sjoerd ==> 4127

**It shows that** Passing an integer after the ':' will cause that field to be a minimum number of characters wide. This is useful for making tables pretty

**It is for string. But, for the number the result is quite the opposite.**

**Test**

print('{:10}'.format(1))

**Result**

>>> 1

## Reading and Writing Files

???with statement

??? An *expression-less* except clause, if present, must be last; it matches any exception.

???[no files need to practice] Methods of File Objects

??? In statement open(file, mode), when the mode is ‘w’, how to understand “truncating the file if it already exists”

??? json.dump(x, f)

after this statement, how to read the file.

???what is the error of the following snippet

ll=[1,'simple','list']

json.dump(ll,f)

ll=json.load(f)

# 8 Errors and Exceptions

## Handling exceptions

??? x = int(input("Please enter a number: "))

when enter non-integer number after executing this line of code, ValueError would be shown by the system as this:

“ValueError: invalid literal for int() with base 10: '9.9'”

But there is no exception after executing code ”int(9.9)”

# Problemset

【built-in】It presents a prompt to the user (the optional arg of raw\_input([arg])), gets input from the user and returns the data input by the user in a string. See the docs for raw\_input().This differs from input() in that the latter tries to interpret the input given by the user; it is usually best to avoid input() and to stick with raw\_input() and custom parsing/conversion code.

【Slicing a dictionary】

You should be iterating over the tuple and checking if the key is in the dict not the other way around, if you don't check if the key exists and it is not in the dict you are going to get a key error:

print({k:d[k] for k in l if k in d})

Some timings:

{k:d[k] for k in set(d).intersection(l)}

In [22]: %%timeit

l = xrange(100000)

{k:d[k] for k in l}

....:

100 loops, best of 3: 11.5 ms per loop

In [23]: %%timeit

l = xrange(100000)

{k:d[k] for k in set(d).intersection(l)}

....:

10 loops, best of 3: 20.4 ms per loop

In [24]: %%timeit

l = xrange(100000)

l = set(l)

{key: d[key] for key in d.viewkeys() & l}

....:

10 loops, best of 3: 24.7 ms per

In [25]: %%timeit

l = xrange(100000)

{k:d[k] for k in l if k in d}

....:

100 loops, best of 3: 17.9 ms per loop

I don't see how {k:d[k] for k in l} is not readable or elegant and if all elements are in d then it is pretty efficient.

============