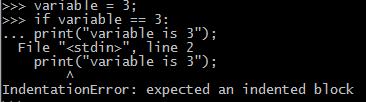
# Python Study

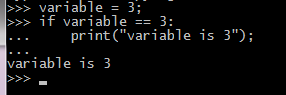
## Turtorial

### Indent

Python is sensitive with spaces and indented block(缩进). So please be very careful when writing codes.

A simple example:

 #need an indented block

 #added an indented block

## Using Python Interpreter

Python use UTF-8 by default, while the standard library uses ASCII character identifiers only.

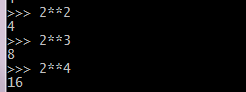
## An Informal Introduction to Python

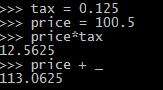
### comment

#Comments in python is started with hash character ‘#’ until the end of a physical line;

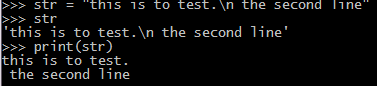
### New operator

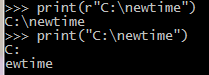
#Division in python always returns a float number; But if you want an integer, you can use // instead of / to get a floor division which returns an integer;

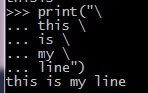
#In python, power is much simpler: \*\* is used to represent power (maybe we could also realize this in C++ with operator override);

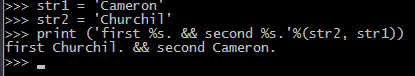
#In interactive mode, the last printed expression is assigned to the variable \_.

### Print()

#print is similar to printf/cout in C/C++;

#use ‘r’ to interpret ‘\’ as normal character;

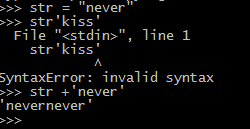
#use \ as connecting symbols(like we use in C++ macro);

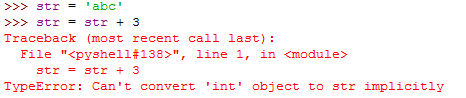
#print can use the old % style. But do remember to put all variables in one %, like this.

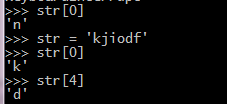


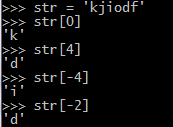
### String

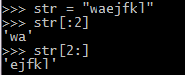
#string can be multipled by number, and be added with string.

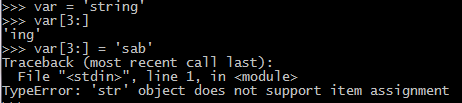
#do remember use + to connect string,

#variable and constant string can’t be connected together;

#string is equal to char[]? Seems.

#python expands great function: a negative index could be located in a string; Negative index starts from -1 not -0, because -0 is viewed equal as 0;

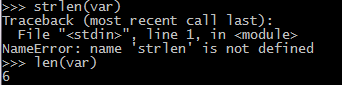
# this is like matlab.

#python string is constant, unmodifiable rvalue;

#string has method **lower().**

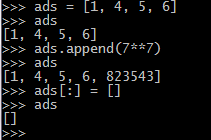
#and also the method **upper().**

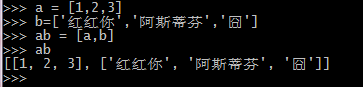
### Len()

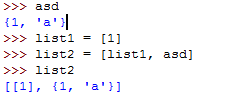
#len() replaces strlen(), or sizeof?

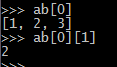
### List

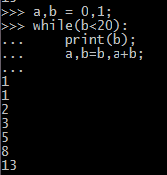
#asd is a list, which can use combound data type;

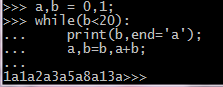
#List can add new members with .append(), and can be cleared with =[];

#List can be nested

#list can be nested with other data types.

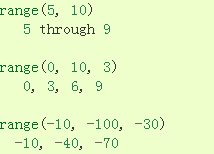
#members can be located as in an array;

#Right-hand side expressions take place first;

#can use end=’’ to illustrate end of print, which is set to ‘\n’ defaultly;

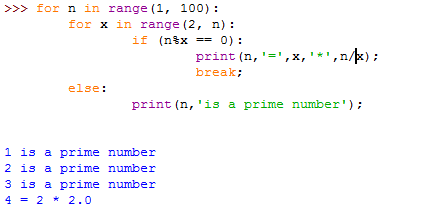
## Control Flow Statements

### Range()

#range can be used

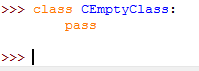
#range can be printed, but the result…

### Else(for loop)

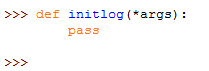
#It’s a great improvement that loop statements in python have an ‘else’ condition, which is satisfied when out of range(for), or becomes false(while).

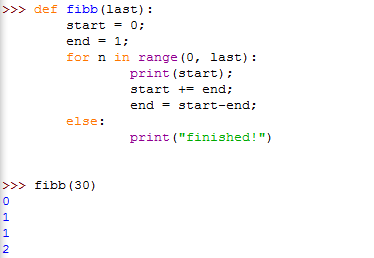
#here I pressed a keyboard. Pass is busy-wait for keyboard interrupt(ctrl+c). Tip: here is a mistake: should’ve used **True** instead of true, but everything in python except 0, is all set to True;

### pass

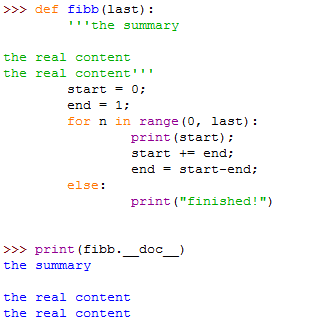
#pass can also be used for creating a minimal class;

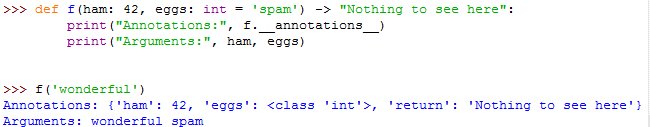
### def

#can also be used as unimplemented words in func;

#**def** keyword introduces a function definition.

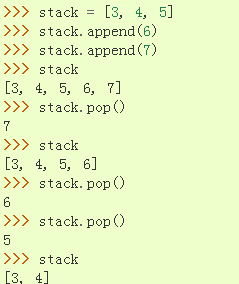
### DocString & annotations

#function documentation, just so so…

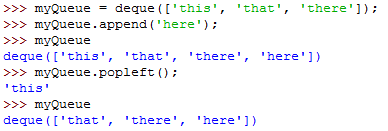
#function annotation, honestly speaking, I’ve not figured out the differences with docString except with parameters.

## Date Structure

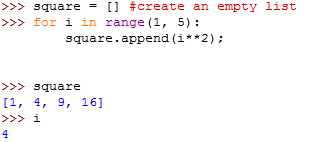
### stack

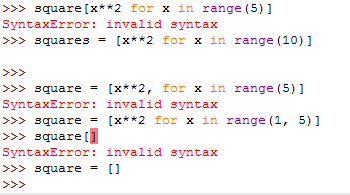
#seems stack is a special kind of List, FILO;

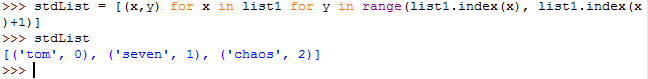
### queue

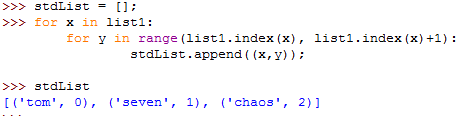
#queue is FIFO, to implement a queue, should import **deque** from **collections** module;

### list

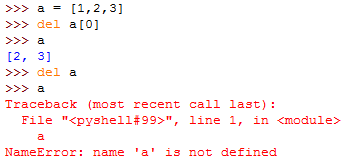
#create an empty List with []…Besides, notice that the variable i always exists. To make it a local variable , we can do the upper like this:

#see? You must use **=[]** to init a List, not a **[]**, cause it can’t be explained as an constructor operator; And a comma symbol should not appear before the **for** loop.

#a senior example of **List Comprehension.** Which is equal to this:



### del

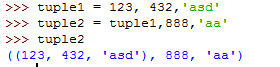
#**del** could be used as delete, to delete members or a whole variable(Wonder if it’s like **delete** critical word in C++?)

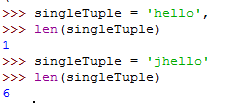
### List & tuple

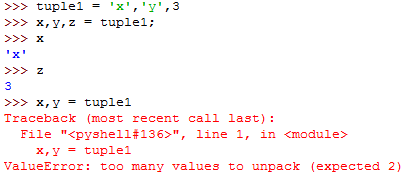
**#List is mutable, tuple is unmutable!!!**

List is often homogeneous accessed via iterator, while tuple is often heterogeneous, accessed via unpacking;

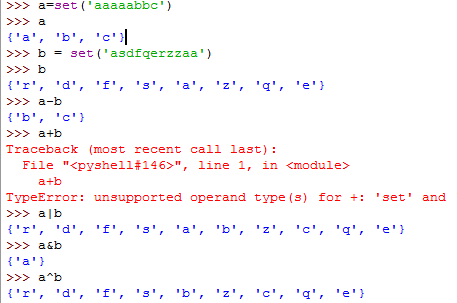
### Tuple

#be aware of the **()** that tuple has.

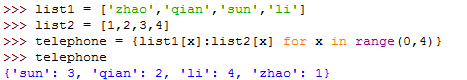
#to create a tuple with only 1 element;

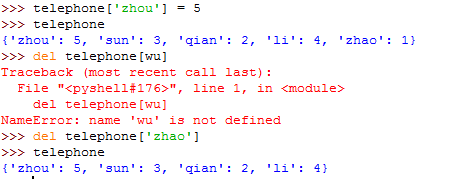
#tuple can be packed and unpacked, but unpack needs appropriate parameters.

### set

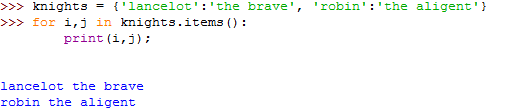
#notice that the init of a set must be **=set(),** different from tuple or list; While the view of a set is contained with **{} not ()**.#basic operators of –(in a but not in b), |(in a or in b), &(in a while in b), ^(in a or in b but not both)

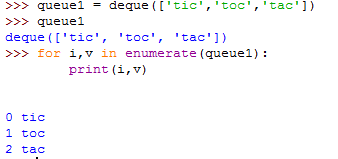
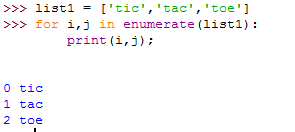
### Dictionary

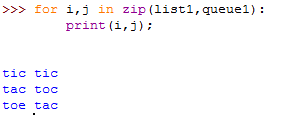
#dictionary is **unordered!**

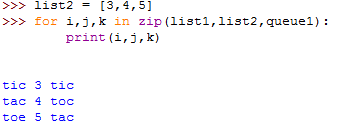
#add new members with simple statement, delete members with **del** critical word;

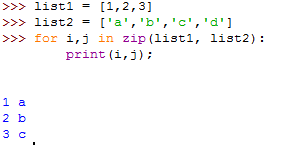
### Looping techniques

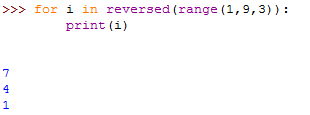
#dictionary key:value can be returned at the same time using items();

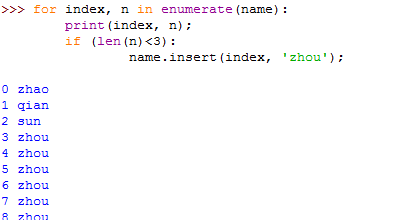
#looping through an sequence(ordered structure such as list or deque), the index and value can be retrieved at the same time using enumerate();

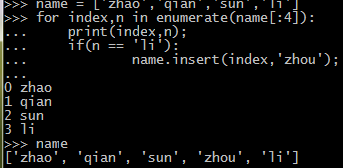


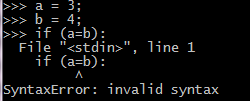
#use **zip()** to pair two or more sequences.

#unbalanced sequences can also be paired

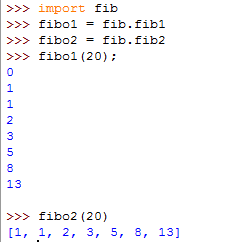
#loop in reverse, the critical word is **reversed()** not reverse().

#always remember to make a copy of original sequence in loop statement, or else will be like upper. The following is recommended:



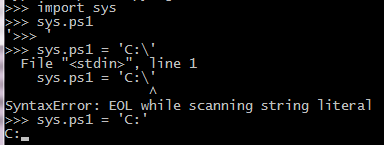
#in python, value can not be assigned within an expression, avoiding the common error: typing a = while intended a ==;

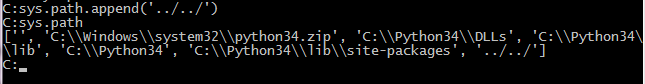
## Modules

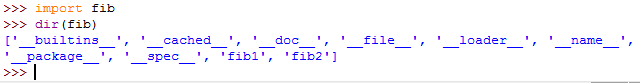
#the imported module should be under sysPath or current directory(on my PC it’s C:\\python34\\). You can edit the module with pythonIDE or Notepad++ (I recommend this).

#can also import like this

#and this, to import all names **except those beginning with an underscore (\_)**. Don’t use this randomly cause it introduces an unknown set of names into the interpreter, possibly hiding sth you’ve already defined;

#can import modules

#sys.path is a list of strings. Which is initialized to a default path, and which can be modified.

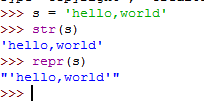
#built-in function **dir()** is used to find out which names a module defines, including variables, modules, functions, etc.

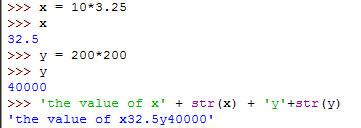


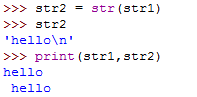
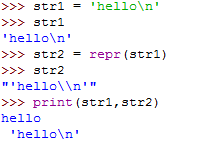
#**dir()** doesn’t list the names of built-in functions and variables.

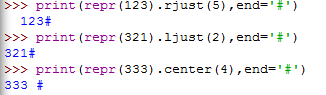
## Input & Output

### Basic string methods

#use **str()** or **repr()** to convert any value to a string;

#change number to string;

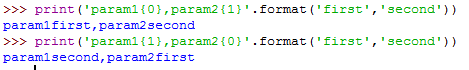
#**str()** is different from **repr()** as repr() always try to explain the parameters to the interpreter, while str() always try to explain the parameter to a string which human easily understands.

#string.rjust(), ljust(), center()

#fill the string left with ‘0’

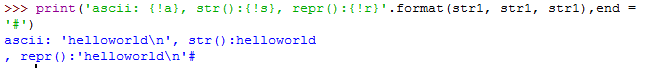
### Str.format()

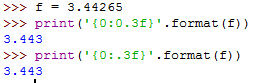
#only {} is useful in str.format(), not [] or ();

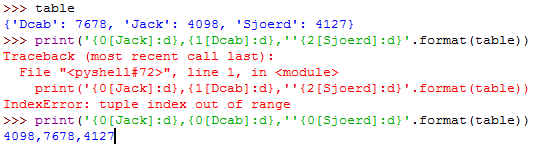
#params in {}

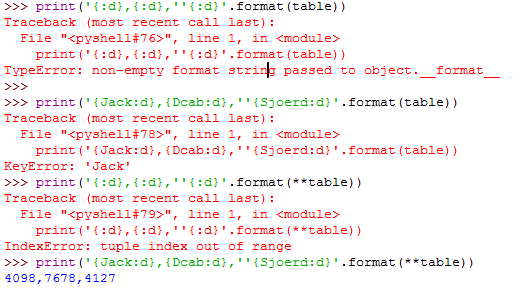
Indicates the index of the parameters in format();

#params can also be indexed with their corres names;

#use !a !s !r to represent **ascii, str(), repr()**;

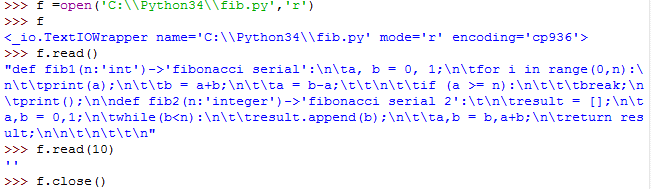
#format

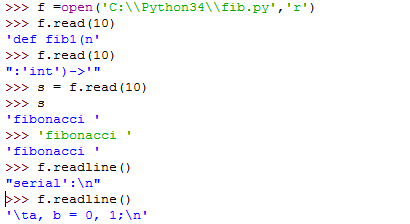
#always remember the meaning of the index is the position of the parameter in **format();** And we can reference the var by name instead of index;

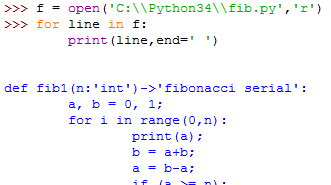
#only the last format is right. Here \*\*table is used as keyword argument;

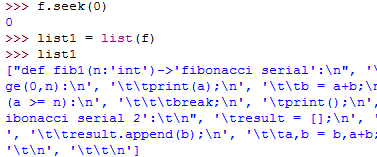
#old printf style, but it seems only one %param is supported.

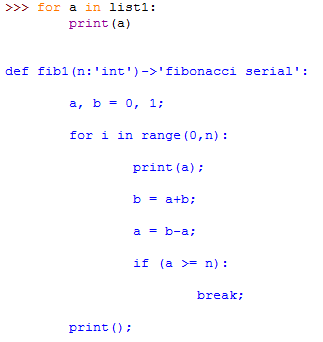
### File operations

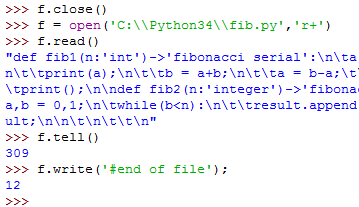
#file contains **open(), read(), close(), readline().**

#readline() ends with \n, if readline() returns an empty string, the end of the file is reached.

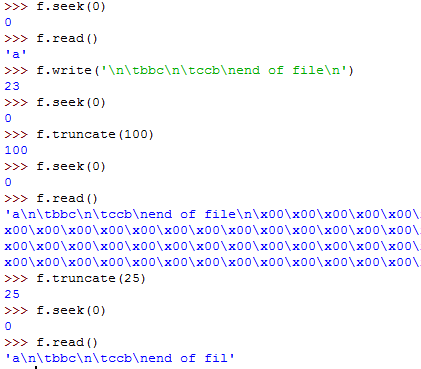
#you can loop over the file object for reading lines from a file. This is memory efficient, fast, and leads to simple code;(**Why??**)

#**list()** is a function which returns all the lines of a file in a list.

#**list()** returns a whole line as a member, like r**eadline()**;

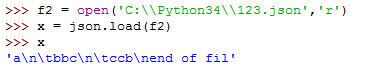
#**tell()** and **write()**

#**with** keyword is a good practice when dealing with file objects.

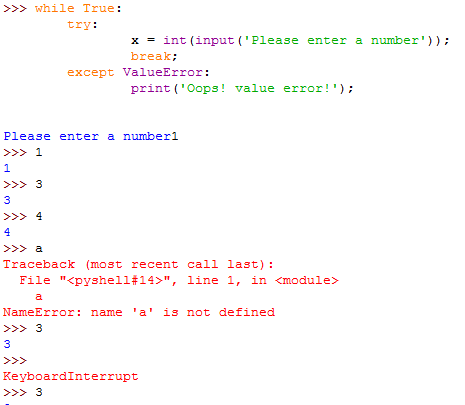
#**truncate()** truncates the file with only the first int(parameters) characters left.

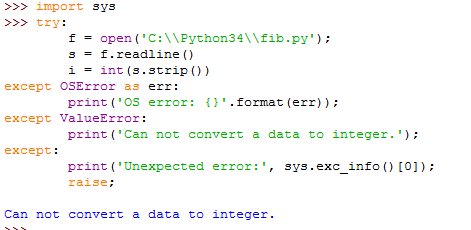
### JSON

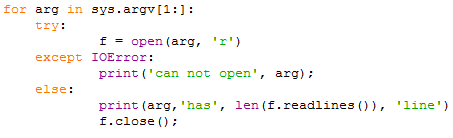
#U can view json string representation of an object with **dumps().**

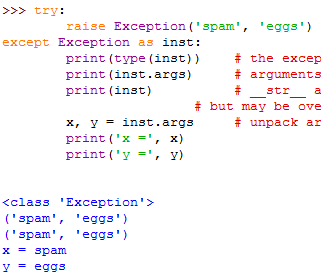
**#**JSON also has **dump()** and **load()** functions in pair to write and read.

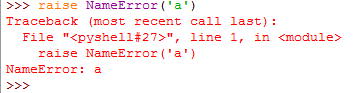
## errors and exceptions

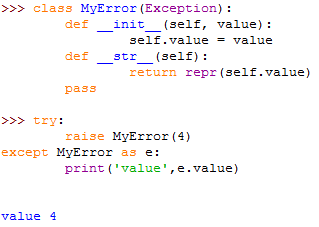
#**try{} except{}** statements

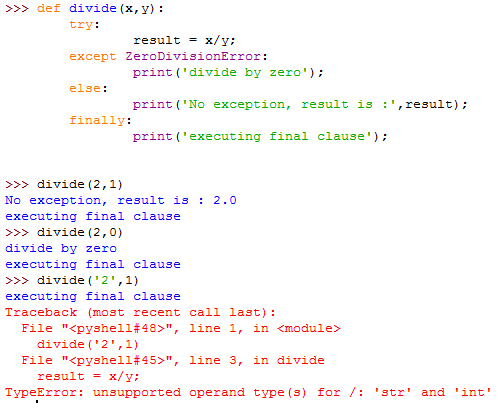
#multi **except**

#**except** has an **else** statement, for code that must be executed if try has no exception.

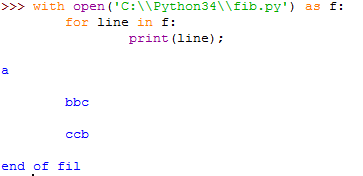
#exception can be organized as a class.

#using **raise**, one can force an exception to happen.

#U can construct a new class which public Exception, and overwrite \_\_init\_\_().

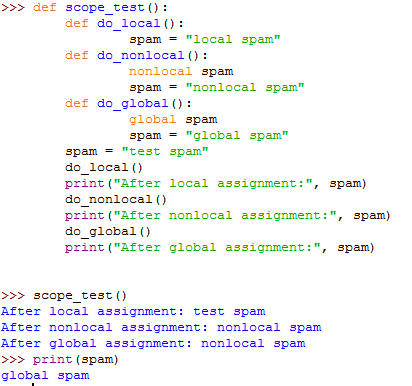
#**else** clause will be executed when no Exception error occurs, while **finally** clause will be executed under every condition. The **finally** clause is useful for releasing existing resources.

### Predefined clean-up actions

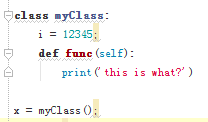
#using **with** as a programmer predefined clean-up action to avoid closing files after operation. It’s very useful in large programs.

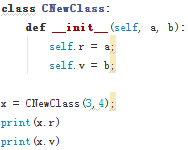
## Classes

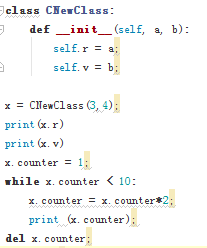
### Scopes and namespaces

#local is the function’s inner var; nonlocal is used to reference outer vars, while global is used to set the var as a global var.

### Class

#class in python is similar to C++, while creating an object pretends that the class object is a parameterless function that returns a new instance of the class.

#creating an instance with **\_\_init\_\_(self).**

#a class instance need not be declared, they spring into existence when they are first assigned to.