## Meeting recordings Notes

### Sep 09, 2021

<https://transunion-my.sharepoint.com/:v:/r/personal/cli_corp_transunion_com/Documents/Recordings/python%20to%20c%20conversion%20training-20210909_142445-Meeting%20Recording.mp4?csf=1&web=1&e=u9OVct>

We only have gradient boost models for now, both skilearn and xgboost are doing the similar things, which are used by our customers, we need those packages to unpack the pkl files and extract the useful information and write up a .c or .h file using Eric’s tool.

Learnt how to load a pkl file;

Learnt what is the features.csv file is, it is a list of attributes being used in this model, please make sure the names in the csv is only attribute names, without algo names;

Learnt what is a pkl file.

Skilearn.py and xgboost.py are the actual script that convert python to C, only for GBM’s for now.

See the model\_to\_c() logic

Tree\_to\_c() is provided by Max, which I don’t know who, need to check with Eric for future support references. The function he provided was for skilearn only. Need to see xgboost as well and check the differences.

Need to look into more about this tree\_to\_c function.

Ask Christine/Eric about the video or material from Max from DSA if it is still available

All material Christine found

R:\\_TEAM\_Christine\Training\PythonTraining\Python\PythonConversion

Pkl is just an extension or you can say a format in python;

you can create any object in Python and write it into a pkl file.

The pkl creation process is like serialization process in Java; what we are going to do is to de-serialize it, or un-pickle it in python if it helps you to understand.

Reference link.

<https://fileinfo.com/extension/pkl#:~:text=A%20PKL%20file%20is%20a,stream%20that%20represents%20the%20objects>.

### Sep 22, 2021

<https://transunion-my.sharepoint.com/:v:/r/personal/cli_corp_transunion_com/Documents/Recordings/python%20to%20c%20conversion%20training-20210922_140823-Meeting%20Recording.mp4?csf=1&web=1&e=5Qc3Du>

in cmd

cd to your working directory.

python g:\

--type [xgboost, sklearn]

--features [.\feature.csv]

--gbm [.\\*.pkl]

--cfile [output.h]

Feature.csv, the order matters.

Because the pkl files are provided by the customer, sometimes the customer may update attribute names or use “labels”, we need to figure out that as well. If we don’t have the information of the expected order of the attributes, we need to figure it out or come back to the customer.

What is the pkl file, and what information are “encapsulated” in the pickle file.

Eric showed how to train a model.

Refer to the TransinSklearn.py file in his folder.

For xgboost object, check the document to find out the properties of this obj.

For example:

>>> pkl.attributes()

{'best\_iteration': '25035', 'best\_ntree\_limit': '25036', 'best\_score': '0.481668'}

for i in range(0,3):

... print(pkl.get\_dump()[i])

...

0:[f11<4.94499969] yes=1,no=2,missing=2

1:[f5<105.785004] yes=3,no=4,missing=4

3:[f14<0.5] yes=5,no=6,missing=6

5:leaf=-0.000246113283

6:leaf=-0.000712651934

4:leaf=0.000113421556

2:leaf=-0.00147898577

0:[f14<2.5] yes=1,no=2,missing=2

1:[f1<1153.5] yes=3,no=4,missing=4

3:[f5<54.5950012] yes=7,no=8,missing=8

7:leaf=-0.000135479408

8:leaf=0.000193596061

4:leaf=-0.000270958815

2:[f17<4.39499998] yes=5,no=6,missing=6

5:leaf=-0.000759136805

6:leaf=-0.00151962193

0:[f14<2.5] yes=1,no=2,missing=2

1:[f5<98.8349991] yes=3,no=4,missing=4

3:[f6<40.4150009] yes=7,no=8,missing=8

7:leaf=-0.000310018455

8:leaf=-6.71494636e-05

4:[f6<98.9799957] yes=9,no=10,missing=10

9:leaf=-0

10:leaf=0.00017857585

2:[f11<5.32499981] yes=5,no=6,missing=6

5:leaf=-0.000751185173

6:leaf=-0.00153669377

### Nov 16, 2021

<https://transunion-my.sharepoint.com/:v:/r/personal/cli_corp_transunion_com/Documents/Recordings/Python%20to%20C%20conversion%20-%20continues-20211116_111611-Meeting%20Recording.mp4?csf=1&web=1&e=9aTBhe>

how to debug python code:

python -m pdb <script name>.py

waiting for Eric’s instruction on how to debug, or maybe I can do some research.

### Nov 18, 2021

#### Part I

<https://transunion-my.sharepoint.com/:v:/r/personal/cli_corp_transunion_com/Documents/Recordings/Python%20to%20C%20conversion%20and%20attribute%20report%20question-20211118_130624-Meeting%20Recording.mp4?csf=1&web=1&e=vD8bMs>

#### Part II

<https://transunion-my.sharepoint.com/:v:/r/personal/cli_corp_transunion_com/Documents/Recordings/Python%20to%20C%20conversion%20and%20attribute%20report%20question%20part%20II-20211118_150525-Meeting%20Recording.mp4?csf=1&web=1&e=cjDdIa>

### Nov 22, 2021

<https://transunion-my.sharepoint.com/:v:/r/personal/cli_corp_transunion_com/Documents/Recordings/R%20conversion%20code%20walkthrough-20211122_133618-Meeting%20Recording.mp4?csf=1&web=1&e=WbnOWT>

### Nov 23, 2021

MoneyLoin AMYL06 conversion from Json to C

<https://transunion-my.sharepoint.com/:v:/r/personal/cli_corp_transunion_com/Documents/Recordings/Json%20to%20R%20discussion-20211123_133241-Meeting%20Recording.mp4?csf=1&web=1&e=xXLvOw>

Ebureau model. I believe they are using a web app to calculate the score, because json data can be used transferred via Http request.

Customer created the answer key and we compare against theirs.

Put all the chaining model DPY’s in the metadata folder for the ADT update.

generatedCppCode.py, update the fname, metadata\_dir, spec\_mapping\_file, what is the output file and results

Refer to jinja package in python for rendering.

How to implement Cpp null in AMYL06

In the generated files.

n\_int32 is a type defined in Eric’s script, which will avoid using present\_flag.

1. Replace all int32\_t with n\_int32, and double with n\_double
2. In var\_data[],

instead of using VAR\_LONGX(AMYL06,br24s , FMT\_NORMAL, '0',chainAttr.br24s),

use VAR\_LONGX(AMYL06,br24s , FMT\_NORMAL, '0',chainAttr.br24s.value\_or(0)),

instead of using VAR\_DOUBLEX(AMYL06,linkf132, 2, FMT\_NORMAL, '0',chainAttr.linkf132),

use VAR\_DOUBLEX(AMYL06,linkf132, 2, FMT\_NORMAL, '0',chainAttr.linkf132.value\_or(0.0)),

1. After var\_data, the null check,

instead of using if (aadm29p->linkf193\_Present[0] != '?'),

use if (chainAttr.linkf193 != nullopt)

Customer only gives us pkl file

We need to create the features.csv file which contains the attribute names.

Check

G:\pyutils\tu\ml\gbm\codeconv\c\sklearn.py and

G:\pyutils\tu\ml\gbm\codeconv\c\xgboost.py to see the logic in there, try to understand it

## How to install packages for python

pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org pandas

In cmd window

If you do not have pandas, run the following command to install

pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org pandas

pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org jinja2

pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org matplotlib

pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org scikit-learn

pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org xgboost

or if you want to install several packages at one time:

pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org pandas jinja2 matplotlib scikit-learn xgboost matplotlib pyxlsb

import pandas as pd

import numpy as np

df=pd.read\_csv("amyl06\_input\_attributes\_OUTPUTFINAL\_20210507.txt",sep="\t")

df.set\_index("key",inplace=True)

df[["CV\_scaledScore","CV\_rawScore"]].to\_csv("answerKey.csv",sep=",")

## How to extract useful information from a customer provided file.

The file we received from the customer is the model they already trained.

The pkl file is either an xgboost object or an skilearn object.

By calling the \_\_write\_skilearn() or \_\_write\_xgboost() function, we will output the C code for that object. The following command will call the function and convert it to c code, so you don’t need to call it manually.

(Future enhancement, we should be able to make it automated, just need more info or more models to figure out the pattern)

In the file customer provided, a dictionary structure contains keys and values.

The keys will be model and dependencies.

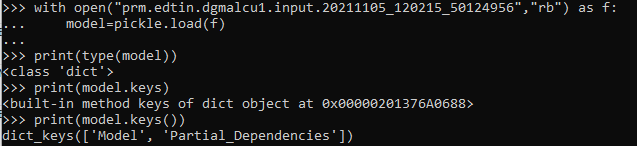
Model is the pkl they trained and want us to convert it into C

Dependencies is the attribute list used for this pkl file.

### 1. Load the file provided by the customer, find out what is in there.

Model is where the pkl resides;

Partial\_Dependencies is the attributes used in this model.



### 2. Extract the pkl file and get the type

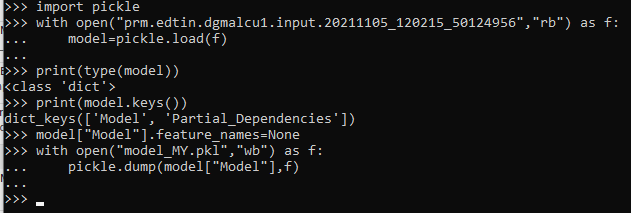
Refer to G:\pyutils\tu\ml\gbm\codeconv\c\skilearn.py

def load\_model(filename):

with open(filename, 'rb') as f:

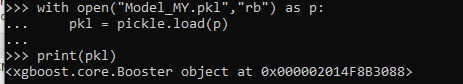
model = pickle.load(f)

return model



You will see a pkl file created in your working directory.

Load the extracted pkl file and get the type



### 3. Extract the features

After retrieving pkl and type, we also need to know what attributes being used in this model.



What you need to do is to grab all the attribute names, without algo name, and convert them in a csv file separated by new line.

Remember to send it to the customer to verify the order.

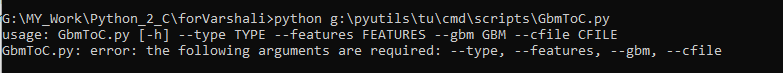
Maybe we have the ability to figure out ourselves.

By calling model.get\_fscore(), which will return a dictionary of the name of attributes and importance.

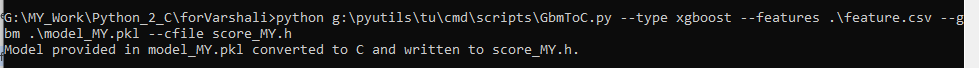
## How to de-serialize the pkl file, extract the logic, and write it out in c

Eric created a script

Usage:



Example:



Command:

G:\MY\_Work\Python\_2\_C\forVarshali>python g:\pyutils\tu\cmd\scripts\GbmToC.py --type xgboost --features .\feature.csv --gbm .\model\_MY.pkl --cfile score\_MY.h

Model provided in model\_MY.pkl converted to C and written to score\_MY.h.

## How to run a GBM model to generate the answer key

Customer will provide a python script to run the model with input.

What we need to do is to change the parameter names and prepare the input data.

## Xgboost.py explanation

To understand this script, you need to understand xgboost model, and have basic understanding of the module pickle.

Refer to xgboost, pickle

#### Modules:

os, pickle, json, xgboost, math, Environment and PackageLoader from jinja2

#### Functions:

load\_model()

Load the model object provided by the customer and return the model instance

load\_and\_convert\_model()

Load the model instance from load\_model().

And passes it to model\_to\_c()

\_\_GetBooster()

Check if model is xgb.Booster,

Return a xgb.Booster instance.

\_\_GetBoosterParams()

Return booster parameters, save\_config() function will return a json string, which can be loaded using json.loads(), which will return a dictionary and can be passed to \_\_GetPredictionParams()

\_\_GetLearnerParams()

Refer to <https://xgboost.readthedocs.io/en/latest/tutorials/saving_model.html>

One drawback is, the output from pickle is not a stable serialization format and doesn’t work on different Python version nor XGBoost version, not to mention different language environments.

So in this function, we may need a guard to check the version.

Need to make sure the version is 1.4.2

>>> parameter["version"]

[1, 4, 2]

Get learner parameters from the config file converted from json to dictionary.

\_\_GetPredictionParams()

Return a tuple of base\_score

model\_to\_c()

Preparation:

Load the config, but I have not figured out how Eric figured out the schema.

Extraction:

This function dumps the model in json format, using json.loads() we can convert the json string to dictionary, which will be easier to work on. This will get us the tree logic.

Conversion:

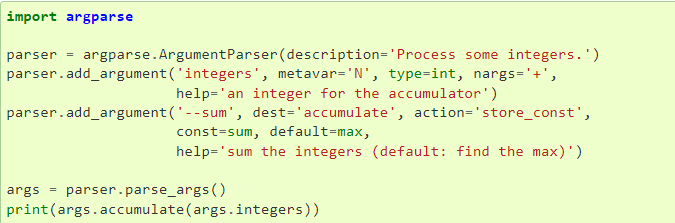
Calls tree\_to\_c()

## Skilearn.py explanation

## GbmToC.py explanation

In order to allow user to pass arguments into the script, and allow the script to take user input as arguments, we need to import argparse

Refer to https://docs.python.org/3/library/argparse.html



class argparse.**ArgumentParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter\_class=argparse.HelpFormatter*, *prefix\_chars='-'*, *fromfile\_prefix\_chars=None*, *argument\_default=None*, *conflict\_handler='error'*, *add\_help=True*, *allow\_abbrev=True*, *exit\_on\_error=True*)

First thing to do is to create an object of ArgumentParser.

parser = argparse.ArgumentParser()

## Python Tutorial

### Syntax

#### Indentation

Unlike Java or other language, indentation in python is important to indicate a block of code.

It doesn’t matter how many blank spaces you put in there, as long as there is at least one.

#### Variables

Python is not a strict type of language as Java.

Whatever value you assign to a variable, python will automatically detect that. Like var in Java, but not generic data type.

### Comments

#### One line comment:

starts with #

#### Multiline comments :

python doesn’t support multiline comments

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

### Variables

#### Creating a variable

Variables do not need to be declared with any particular type, and can even change type after they have been set.

#### Variable names

Camel case: myVariableName

Pascal case: MyVariableName

Snake case: my\_variable\_name

#### Assign variables

1. Multiple values to multiple variables

x, y, z = "Orange", "Banana", "Cherry"

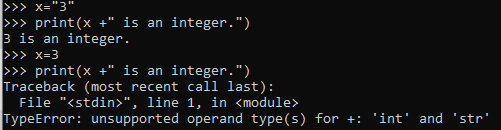
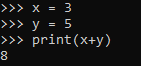
1. One value to multiple variables

x = y = z = "Orange"

#### Output variables

Similar to Java, use + in print().

However, notice you have to use the same data type if you use + .

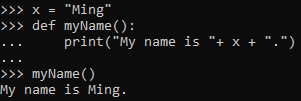
 

You can also use comma, which will allow you to concatenate

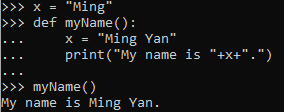
For example



#### Global Variables

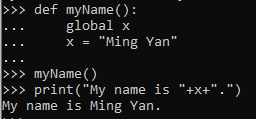


Vs

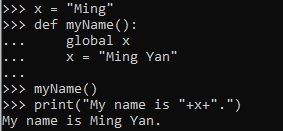


##### Global keywords

If you use the global keyword, the variable belongs to the global scope:



In addition, it can be used to update a global variable outside the function.



#### Casting

If you want to specify the data type of a variable, this can be done with casting.

For example:

x = int(3) or x = int(“3”)

y = str(3) or y = str(“3”)

type(x) will give out <class 'int'>

type(y) will give out <class 'str'>

y = 3

type(y) will give out <class ‘int’>

For more information, please refer to x

### Data Types

#### Built-in data types

|  |  |
| --- | --- |
| Text Type: | str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | dict |
| Set Types: | set, frozenset |
| Boolean Type: | bool |
| Binary Types: | bytes, bytearray, memoryview |

#### Getting the data type, type()

You can use the type() to output the data type of a variable.

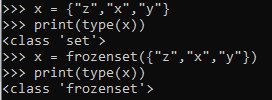
For example:



#### Setting the data type

|  |  |
| --- | --- |
| Data type | example |
| str | x = “Ming Yan” |
| int | x = 36 |
| float | x = 11.29 |
| complex | x = 1j |
| list | x = [“apple”,”banana”,”cherry”] |
| tuple | x = (“apple”,”banana”,”cherry”) |
| set | x = {“apple”,”banana”,”cherry”} |
| frozenset | x = frozenset({“apple”,”banana”,”cherry”}) |
| range | x = range(6) |
| dict | x = {“Ming”: 36, “Sr Analyst”: 3} |
| bool | x = True |
| bytes | x = b”Ming” |
| bytearray | x = bytearray(5) |
| memoryview | x = memeoryview(bytes(5)) |

For example:

, 

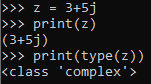
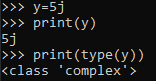
#### Number:

##### Int type: The same as all other languages

##### Float type: can also be scientific number with an e to indicate the power of 10.



##### Complex type: complex numbers are written with a “j” as the imaginary part.

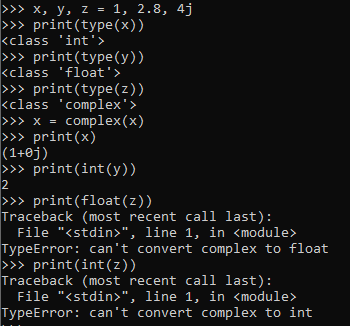
 

##### Type Conversion:

You can convert from one type to another with int(), float(), and complex() methods:

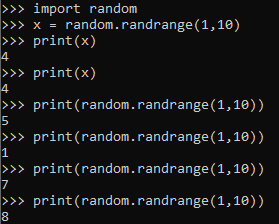
Notice you cannot convert complex to either int or float

However, you can convert int and float to complex.



##### Random number:

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:



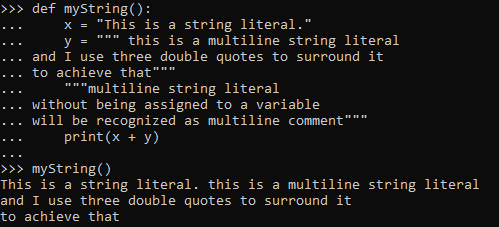
#### String:

Strings in python are surrounded by either single quotation marks or double quotation marks.

Multiline Strings starts with “”” or ‘’’.

However, python only recognize multiline strings being assigned to a variable. For those multiline strings without being assigned to a variable, python will ignore it and see it as multiline comments.

For example:



##### Strings Manipulations

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

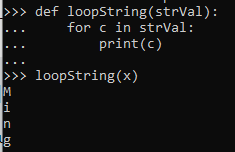
Square brackets can be used to access elements of the string.



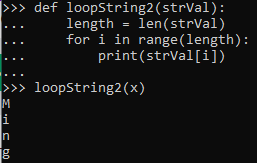
Like Java, strings in python are also immutable, which means all string methods returns new values. They do not change the original string.

###### Loop through a String

Given the truth of a string is actually an array, we can loop through it.



Or you can check the length of a string and loop thru it, using len() to check the length



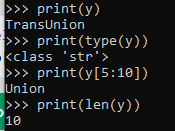
###### Slice a string or sub string

Given the truth that strings are actually arrays, we can do many things with strings by treating them as arrays.

TransUnion

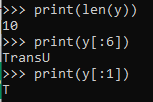
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | r | a | n | s | U | n | i | o | n |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Slicing by specify the starting position and the ending position (not included)



Slicing from the start

By leaving out the start index, the range will start at the first character:



Slicing to the end

By leaving out the end index, the rang will go to the end:



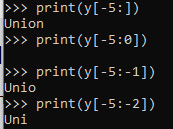
Negative indexing

Use negative indexes to start the slice from the end of the string:

Notice you cannot specify 0 at the end to get the substring starting from the negative index.

TransUnion

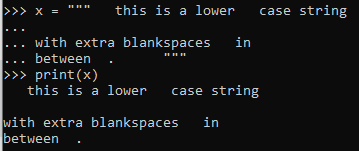
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| T | r | a | n | s | U | n | i | o | n |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |



###### Modify a string

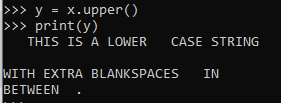
Python has a set of build-in methods that you can use on strings.

Use this as an example.



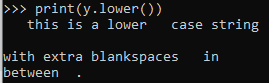
Upper case

upper()



Lower case

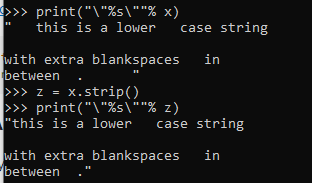
lower()



Remove white space

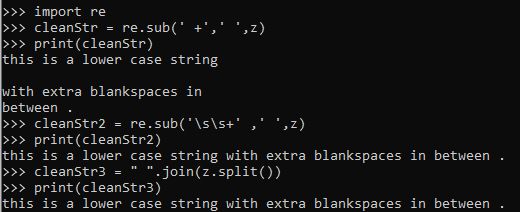
strip() will only remove leading and trailing blanks for a string.

Blanks in between will not be removed by calling this function.



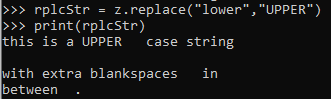
In order to remove all extra blank spaces in a string.

But notice the last one, before the dot, I will update it later.



Replace string

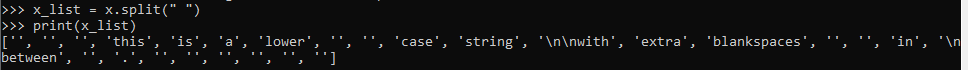
replace(str1, str2), replace str1 with str2





Split string

The split() method returns a list where the text between the specified separator becomes the list items.



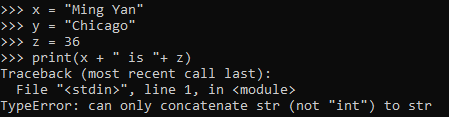
String concatenation

To concatenate, or combine, two strings you can use the + operator, like Java.

Notice only string of the same type can be “added” together.

###### Format strings

As we learn in Python variables chapter, we cannot combine strings and numbers like this:



But we can combine strings and numbers by

1. casting that int to string



2. Using % Operator



3. using the format() method.



Or



Or using index inside the {}



4. using f-strings (after python 3.6)

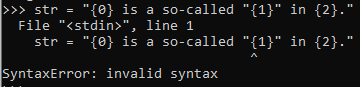


###### Escape characters

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

For example, if you want to output double quotes inside a string



By using the escape character “\”



Or using single quotes instead of double quotes

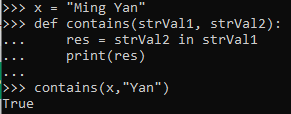


|  |  |
| --- | --- |
| code | result |
| \’ | single quote |
| \” | double quote |
| \\ | backslash |
| \n | new line |
| \r | carriage return |
| \t | tab |
| \b | backspace |
| \f | form feed |
| \ooo | Octal value |
| \xhh | Hex value |

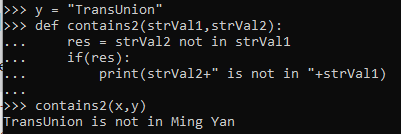
###### Check if a string contains a certain substring

We don’t have contains()in Python like Java, but python offer a more intuitive way to check.

Keyword in



Also, you can use not in

****

##### String Methods

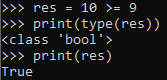
Learn more about String Methods with our [String Methods Reference](https://docs.python.org/3/library/stdtypes.html#string-methods)

#### Boolean:

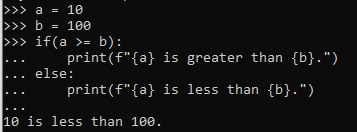
Booleans represent one of two values: True or False.

Any expression in python will get a Boolean value.

For example:

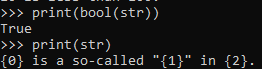
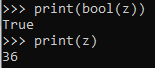


Or



##### Evaluate values and variables

Function bool() allows you to evaluate any value, and return True or False.

###### Most Values are True

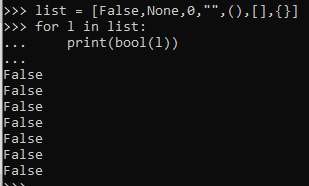
Almost any value is evaluated to True if it has some content.

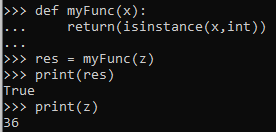
Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

###### Some Values are False





#### Operators:

Python divides the operators in the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Identity operators
* Membership operators
* Bitwise operators

##### Arithmetic operators:

Arithmetic operators are used with numeric values to perform common mathematical operations:



|  |  |  |
| --- | --- | --- |
| operator | Name | Example |
| + | addition |  |
| - | subtraction |  |
| \* | multiplication |  |
| / | division |  |
| % | modulus |  |
| \*\* | exponentiation |  |
| // | floor division |  |

##### Bitwise operators:

|  |  |  |
| --- | --- | --- |
| operator | name | description |
| & | and | Sets each bit to 1 if both bits are 1 |
| | | or | Sets each bit to 1 if one of two bits is 1 |
| ^ | xor | Sets each bit to 1 if only one of two bits is 1 |
| ~ | not | Inverts all the bits |
| << | zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

##### Assignment operators:

Almost the same as Java

Assignment operators are used to assign values to variables:

|  |  |  |
| --- | --- | --- |
| operator | Example | Equivalent to |
| = |  | x = 5 |
| += |  | x = x + 3 |
| -= |  | x = x - 3 |
| \*= |  | x = x \* 3 |
| /= |  | x = x / 3 |
| %= |  | x = x % 3 |
| //= |  | x = x // 3 |
| \*\*= |  | x = x \*\* 3 |
| &= |  | x = x & 3 |
| |= |  | x = x | 3 |
| ^= |  | x = x ^ 3 |
| >>= |  | x = x >> 3 |
| <<= |  | x = x << 3 |

##### Comparison operators:

Comparison operators are used to compare two values:



|  |  |  |
| --- | --- | --- |
| operator | Name | example |
| == | equal |  |
| != | not equal |  |
| > | greater than |  |
| < | less than |  |
| >= | greater than or equal to |  |
| <= | less than or equal to |  |

== doesn’t work the same as operator is

##### Logical operators:

Logical operators are used to combine conditional statements:



|  |  |  |
| --- | --- | --- |
| operator | description | example |
| and | return True is both are true |  |
| or | returns True if one of the statements is true |  |
| not | Reverse the result, returns False if the result is true |  |

##### Identity operators:

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:



|  |  |  |
| --- | --- | --- |
| operator | description | example |
| is | Returns True if both variables are the same object |  |
| is not | Returns True if both variables are not the same object |



#### Array:

Python does not have built-in support for Array data type, but List can be used instead.

However, to work with arrays in Python you will have to import a library, from array import \*

This module defines an object type which can compactly represent an array of basic values: characters, integers, floating point numbers.

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together.

class array.array(typecode[, initializer])

A new array whose items are restricted by typecode, and initialized from the **optional** initializer value, which must be a list, a bytes-like object, or iterable over elements of the appropriate type.

If given a list or string, the initializer is passed to the new array’s fromlist(), frombytes(), or fromunicode() method (see below) to add initial items to the array. Otherwise, the iterable initializer is passed to the extend() method.

Raises an auditing event array.\_\_new\_\_ with arguments typecode, initializer.

| **Type code** | **C Type** | **Python Type** | **Minimum size in bytes** | **Notes** |
| --- | --- | --- | --- | --- |
| 'b' | signed char | int | 1 |  |
| 'B' | unsigned char | int | 1 |  |
| 'u' | wchar\_t | Unicode character | 2 | (1) |
| 'h' | signed short | int | 2 |  |
| 'H' | unsigned short | int | 2 |  |
| 'i' | signed int | int | 2 |  |
| 'I' | unsigned int | int | 2 |  |
| 'l' | signed long | int | 4 |  |
| 'L' | unsigned long | int | 4 |  |
| 'q' | signed long long | int | 8 |  |
| 'Q' | unsigned long long | int | 8 |  |
| 'f' | float | float | 4 |  |
| 'd' | double | float | 8 |  |

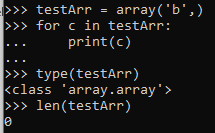
It can be 16 bits or 32 bits depending on the platform.

Changed in version 3.9: array('u') now uses wchar\_t as C type instead of deprecated Py\_UNICODE. This change doesn’t affect to its behavior because Py\_UNICODE is alias of wchar\_t since Python 3.3.

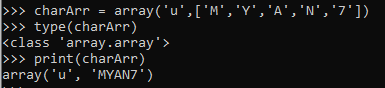
Deprecated since version 3.3, will be removed in version 4.0.

For example:

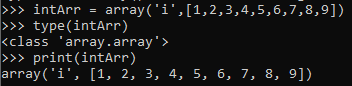
List of initializer is optional:



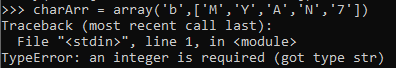
##### char array:

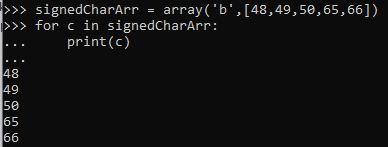


##### int array:



##### Signed char array:





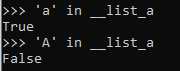
You get the idea I believe.

##### Array methods:

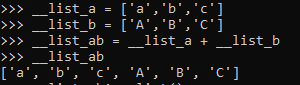
Refer to [Python.API.array](https://docs.python.org/3/library/array.html)

#### Common Sequence Operations

in or not in



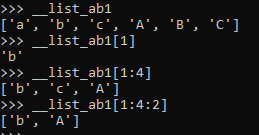
s + t



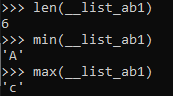
s\*n or n\*s



s[i] , s[i:j], s[i:j:k]

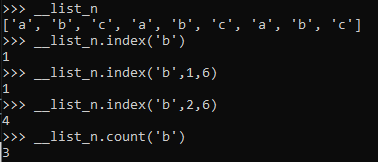


len(s), min(s), max(s)



s.index(x[,i[,j]]) index of the first occurrence of x in s (at or after index I and before index j)

s.count(x)



#### Mutable Sequence Operations

Tested yourself. I already know that stuff.

s[i] = x, s[i:j] = t, s[i:j:k] = t

del s[i:j], del s[i:j:k]

s.append(x)

s.clear()

s.copy()

s.extend(t) or s += t

s \*= n

s.insert(i,x)

s.pop() or s.pop(i)

s.remove(x)

s.reverse()

1. t must have the same length as the slice it is replacing.
2. The optional argument i defaults to -1, so that by default the last item is removed and returned.
3. remove() raises ValueError when x is not found in s.
4. The reverse() method modifies the sequence in place for economy of space when reversing a large sequence. To remind users that it operates by side effect, it does not return the reversed sequence.
5. clear() and copy() are included for consistency with the interfaces of mutable containers that don’t support slicing operations (such as dict and set). copy() is not part of the collections.abc.MutableSequence ABC, but most concrete mutable sequence classes provide it.

New in version 3.3: clear() and copy() methods.

1. The value n is an integer, or an object implementing \_\_index\_\_(). Zero and negative values of n clear the sequence. Items in the sequence are not copied; they are referenced multiple times, as explained for s \* n under Common Sequence Operations.

Lists also provide the following additional method:

sort(\*, key=None, reverse=False) , which is mentioned in Sort a list.

#### List:

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are **mutable sequences**, typically used to store collections of **homogeneous** items (where the precise degree of similarity will vary by application).

class list([iterable])

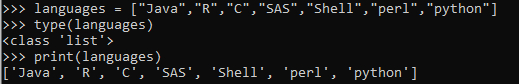
Lists may be constructed in several ways:

* Using a pair of square brackets to denote the empty list: []
* Using square brackets, separating items with commas: [a], [a, b, c]
* Using a list comprehension: [x for x in iterable]
* Using the type constructor: list() or list(iterable)

The constructor builds a list whose items are the same and in the same order as iterable’s items. iterable may be either a sequence, a container that supports iteration, or an iterator object. If iterable is already a list, a copy is made and returned, similar to iterable[:]. For example, list('abc') returns ['a', 'b', 'c'] and list( (1, 2, 3) ) returns [1, 2, 3]. If no argument is given, the constructor creates a new empty list, [].

Here I provided 2 ways a list can be created:

using square brackets:

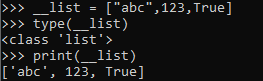


Using constructor:

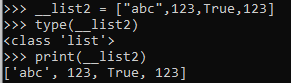


##### Features of a list:

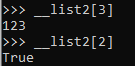
###### Allow mixed types of elements



###### Allow duplicates

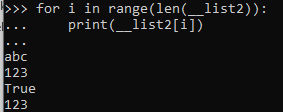


###### Indexed



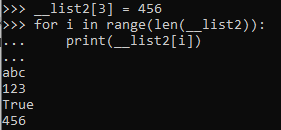
###### Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.



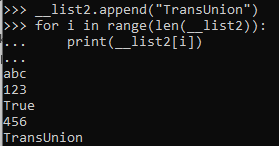
###### Mutable

1. Update element:

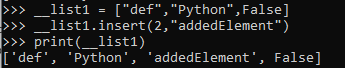


1. Add an element:

Use append()

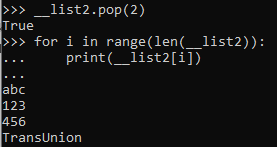


Using insert()

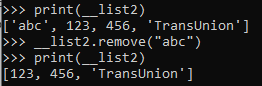


1. Remove an element:

Use pop(), pop() will take index as argument



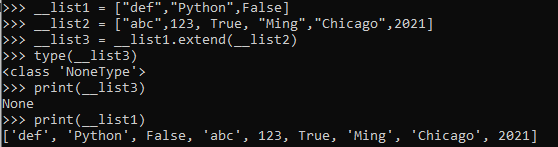
Use remove(), remove() will take value as argument



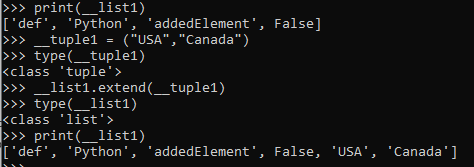
1. Merge 2 lists

Use extend() to merge the list in the argument to the list who invokes the function.

Notice extend() will return void, so you cannot assign a new list to store the result.

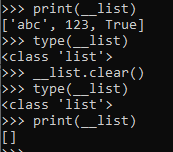


extend() also accepts any other iterable object (tuples, sets, dictionaries etc.)



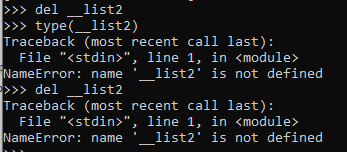
1. Clear a list

Use clear() to empty a list



1. Delete a list

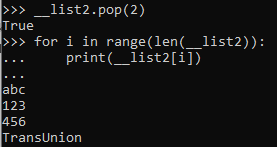
Use keyword del to delete the list completely.



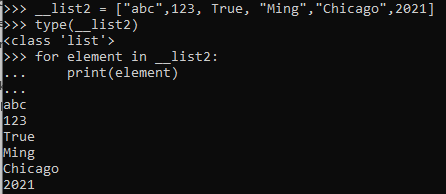
##### Loop through a list

###### For loop

Using index

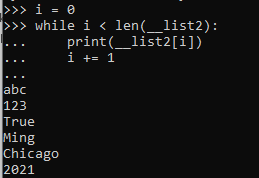


Using enhanced for loop



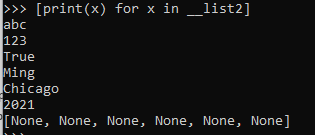
###### While loop

Notice in python there is no i++



###### Looping Using List Comprehension

Not sure why the [None, None, None] stuff at the end.



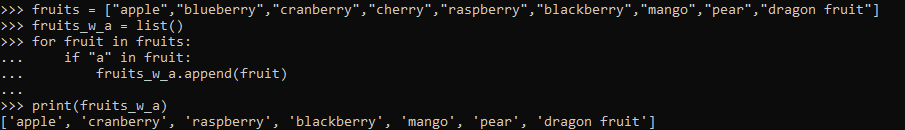
##### List Comparison:

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

For example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension, you will have to write a for statement with a conditional test inside:



With list comprehension, you can do all that with only one line of code:



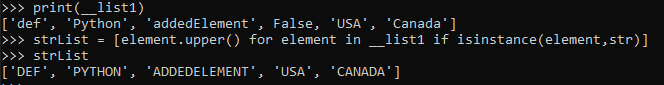
Syntax of list comprehension:

newlist = [expression for item in iterable if condition == True]

The return value is a new list, leaving the old list unchanged.

###### Expression:

The expression is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list



###### Iterable:



##### Sort a list

Lists implement all of the common and mutable sequence operations. Lists also provide the following additional method:

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.

The functools.cmp\_to\_key() utility is available to convert a 2.x style cmp function to a key function.

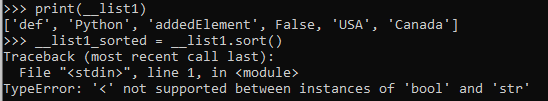
reverse is a bool value. If set to True, then the list elements are sorted as if each comparison were reversed.

This method modifies the sequence in place for economy of space when sorting a large sequence. To remind users that it operates by side effect, it does not return the sorted sequence (use sorted() to explicitly request a new sorted list instance).

The sort() method is guaranteed to be stable. A sort is stable if it guarantees not to change the relative order of elements that compare equal — this is helpful for sorting in multiple passes (for example, sort by department, then by salary grade).

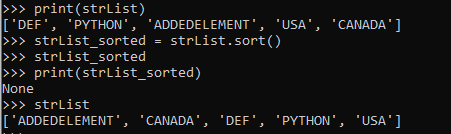
Sort List Alphanumerically

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:



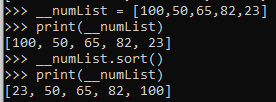
To sort a list, you have to make sure all the elements in that list are the homogenous type.

Remember I had a strList which was created using list comprehension by extracting all the str type elements from \_\_list1?

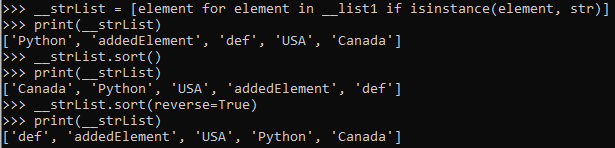


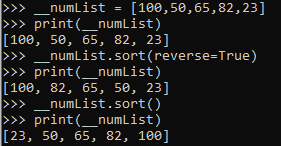
Notice that sort() doesn’t return a list. It modifies the original list.

Sort the list numerically:



Sort a list in descending order, reverse is default to False.



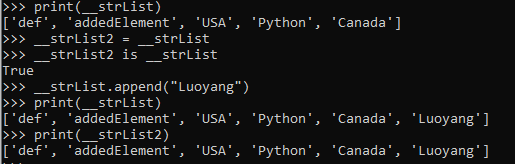


##### Copy a list

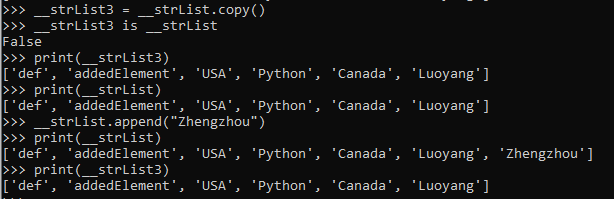
You cannot use = to assign another list

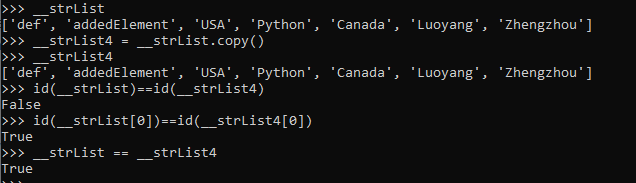


\_\_strList2 is only a reference to \_\_strList, and changes made in \_\_strList will automatically also be made in \_\_strList2.

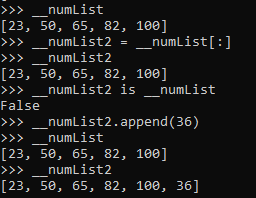


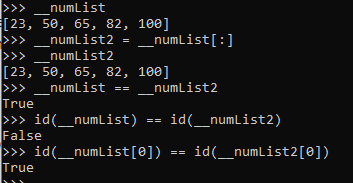
But using copy(), you will have a list with the same element, even you change the original list, the copied list will not be affected.



Because they are in different memory locations:

To achieve a shallow copy of a list, you can also use slicing





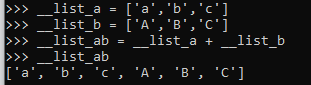
Pending:

Shallow copy vs deep copy

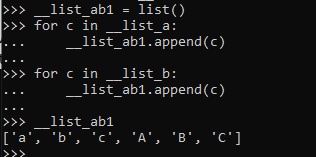
##### Join lists

There are several ways to join, or concatenate, two or more lists in Python.

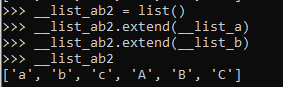
One of the easiest ways are by using the + operator.



Another way to join two lists is by appending all the items from list2 into list1, one by one:



Or you can also use extend() as I mentioned IVMerge 2 lists



##### List methods

Lists implement all of the [common](https://docs.python.org/3/library/stdtypes.html#typesseq-common) and [mutable](https://docs.python.org/3/library/stdtypes.html#typesseq-mutable) sequence operations.

For those functions, refer to hyperlinks above or Common Sequence Operations and Mutable Sequence Operations

#### Tuple:

Tuples are **immutable sequences**, typically used to store collections of **heterogeneous** data (such as the 2-tuples produced by the enumerate() built-in).

Tuples are also used for cases where an immutable sequence of homogeneous data is needed (such as allowing storage in a set or dict instance).

class tuple([iterable])

Tuples may be constructed in a number of ways:

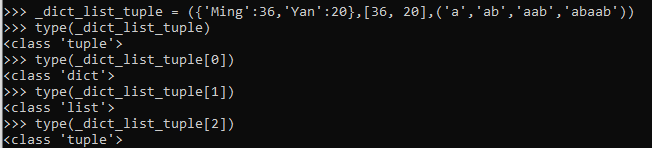
* Using a pair of parentheses to denote the empty tuple: ()
* Using a trailing comma for a singleton tuple: a, or (a,)
* Separating items with commas: a, b, c or (a, b, c)
* Using the tuple() built-in: tuple() or tuple(iterable)

The constructor builds a tuple whose items are the same and in the same order as iterable’s items. iterable may be either a sequence, a container that supports iteration, or an iterator object. If iterable is already a tuple, it is returned unchanged. For example, tuple('abc') returns ('a', 'b', 'c') and tuple( [1, 2, 3] ) returns (1, 2, 3). If no argument is given, the constructor creates a new empty tuple, ().

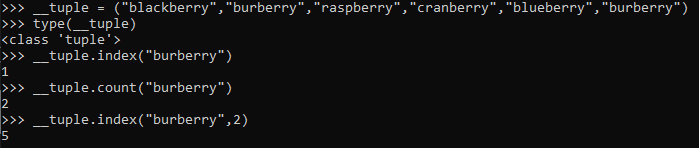
Note that it is actually the comma which makes a tuple, not the parentheses. The parentheses are optional, except in the empty tuple case, or when they are needed to avoid syntactic ambiguity. For example, f(a, b, c) is a function call with three arguments, while f((a, b, c)) is a function call with a 3-tuple as the sole argument.

##### Features of a tuple:

###### Allow mixed types of elements



###### Allow duplicates



###### Indexed

Same as lists

###### Ordered

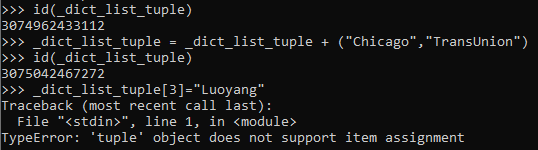
When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Same as lists

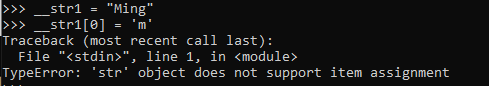
###### Immutable

Tuples are **immutable**, meaning that we cannot change, add or remove items after the tuple has been created.

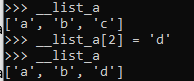
You cannot assign a new value to a tuple.



Same as a string



But you can update a list



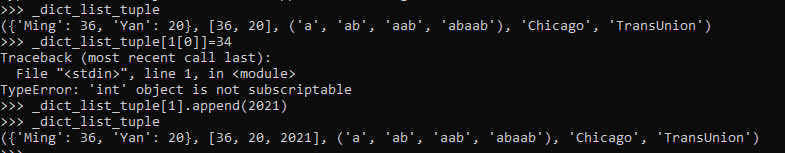
What is the purpose of the immutability of a tuple?

Suppose you write a method that depends on a particular sequence of objects remaining the same throughout the lifetime of the code. If you store these objects in a list, there is a chance that some other method using the list will accidentally alter it and thus break your method. Thus, in a way, a tuple offers a guarantee that a particular collection of objects will remain fixed.

What makes tuple immutable in python?

In Java, final is the keyword that makes string immutable. What makes tuple immutable in python?

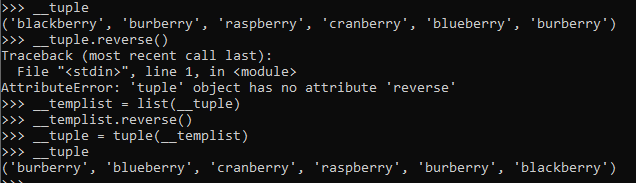
What if immutable tuple contains element that is mutable?



##### Update a tuple

Even though tuples are immutable, remember it is still iterable.

And iterable offers a workaround to update a tuple, which is converting that tuple into a list, update the list, and then convert it back to a tuple.

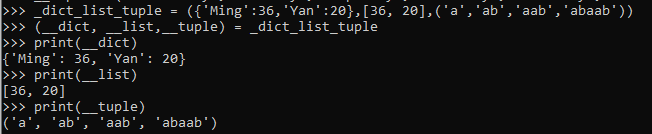


##### Unpack a tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

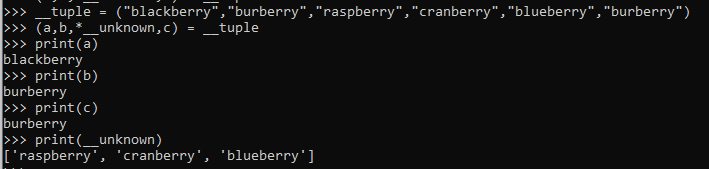


However, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":



Note: The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.





##### Loop through a tuple

Same as lists

##### Join tuples

Tuple extends common sequence, so no extend() for tuple.

##### Tuple methods

Tuples implement all of the [common](https://docs.python.org/3/library/stdtypes.html#typesseq-common) sequence operations.

Refer to Common Sequence Operations

#### Set:

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is **unordered, unchangeable\*, and unindexed**.

\* Note: Set items are unchangeable, but you can **remove items and add new items**.

A set object is an unordered collection of **distinct hashable objects**. Common uses include membership testing, removing duplicates from a sequence, and computing mathematical operations such as intersection, union, difference, and symmetric difference. (For other containers see the built-in dict, list, and tuple classes, and the collections module.)

Like other collections, sets support x in set, len(set), and for x in set. **Being an unordered collection, sets do not record element position or order of insertion**. **Accordingly, sets do not support indexing, slicing, or other sequence-like behavior.**

There are currently two built-in set types, set and frozenset.

**The set type is mutable** — the contents can be changed using methods like add() and remove(). Since it is mutable, it has no hash value and cannot be used as either a dictionary key or as an element of another set.

**The frozenset type is immutable and hashable** — its contents cannot be altered after it is created; it can therefore be used as a dictionary key or as an element of another set.

Non-empty sets (not frozensets) can be created by placing a comma-separated list of elements within braces, for example: {'jack', 'sjoerd'}, in addition to the set constructor.

The constructors for both classes work the same:

class set([iterable])

class frozenset([iterable])

Return a new set or frozenset object whose elements are taken from iterable. The elements of a set must be hashable. **To represent sets of sets, the inner sets must be frozenset objects. If iterable is not specified, a new empty set is returned.**

Sets can be created by several means:

* Use a comma-separated list of elements within braces: {'jack', 'sjoerd'}
* Use a set comprehension: {c for c in 'abracadabra' if c not in 'abc'}
* Use the type constructor: set(), set('foobar'), set(['a', 'b', 'foo'])

##### Features of a set

Unordered

##### Update a set

##### Loop through a set

##### Join sets

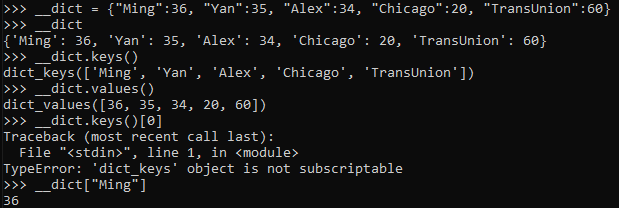
##### Set methods

#### Dictionary:

Dictionary is like Map in Java.

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered\*, changeable and does not allow duplicates.

****

Dictionary items **are ordered**, changeable, and does not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

**Notice**: Unlike sequences, which are indexed by a range of numbers, 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 tuple 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().

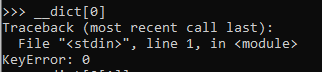
##### Features of a dictionary

###### Ordered

**As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.**

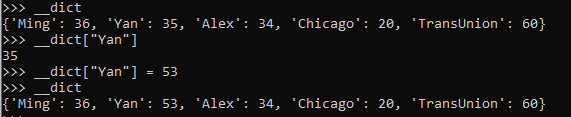
When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

Unordered means that the items does not have a defined order, you cannot refer to an item by using an index. Since dictionary are indexed by keys, so you cannot use index as list or tuples



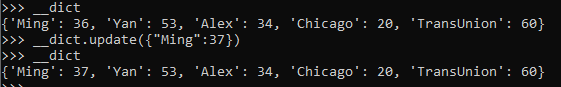
###### Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.



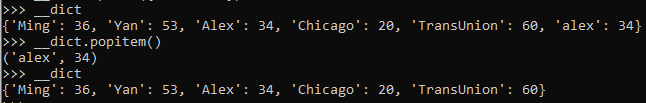
Add/update a key:value pair: update()



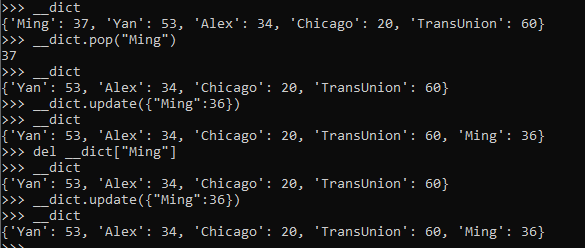




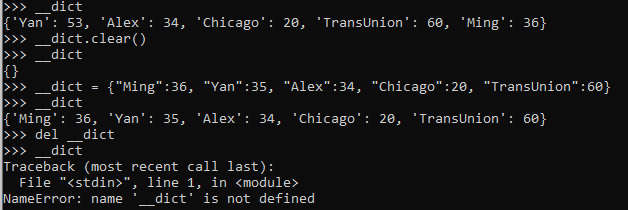
Remove the latest added key:value pair: popitem()



Remove by key:value pair pop() or del



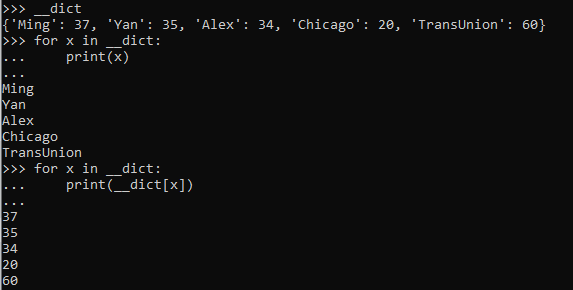
Remove everything: clear() or del



###### No duplicates

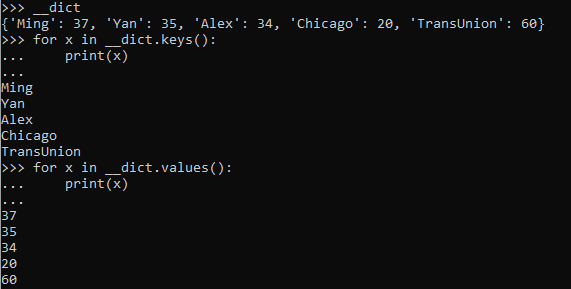


##### Loop through a dictionary

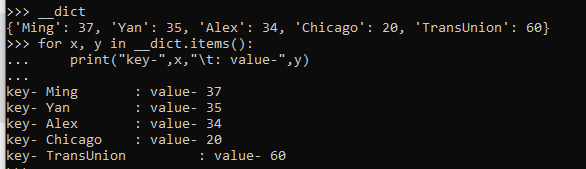


Loop through a dictionary will only loop through the keys as example shown above, which is equivalent to loop through by \_\_dict.keys()

Of course you can loop through the values() too.

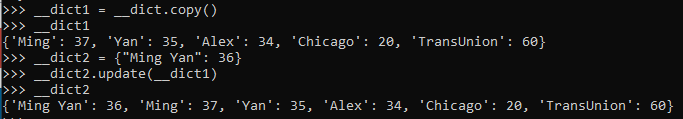


But it doesn’t give me the mapping, so another way to loop through a dictionary is loop thru the item.



##### Join dictionaries update() and a trick

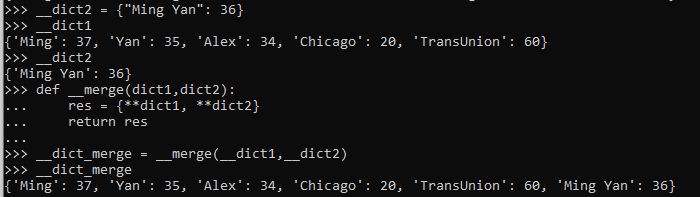
You can call it merge or join, but it boils down to update.



Notice that will update the dictionary who calls update().

But you can use a trick too.

Refer to: <https://www.geeksforgeeks.org/python-merging-two-dictionaries/>



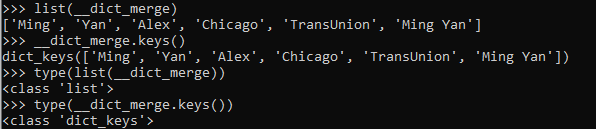
This is generally considered a trick in Python where a single expression is used to merge two dictionaries and stored in a third dictionary. **The single expression is \*\*.** This does not affect the other two dictionaries. \*\* implies that an argument is a dictionary. Using \*\* [double star] is a shortcut that allows you to pass multiple arguments to a function directly using a dictionary. For more information refer \*\*kwargs in Python. Using this we first pass all the elements of the first dictionary into the third one and then pass the second dictionary into the third. **This will replace the duplicate keys of the first dictionary.**

##### Dictionary methods

Refer to [API](https://docs.python.org/3/tutorial/datastructures.html)- 5.5. Dictionaries

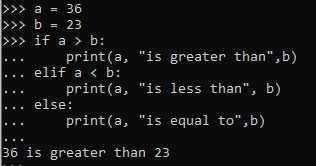
list(dict) will return a list

while dict.keys() will return a dict\_keys



### Conditions

#### if elif else



#### short hand if





#### Nested if

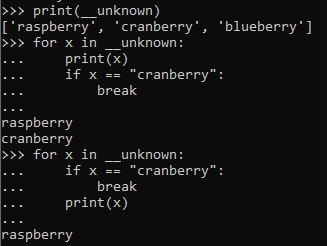
Just remember the identation.

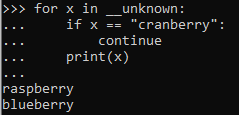
**Note: if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.**

### Loops

#### For loop

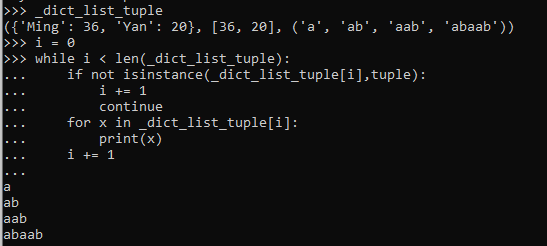
Refer to previous examples.





#### While loop

Refer to previous examples



### Functions

#### Normal function

In python a function is defined using the def keyword:

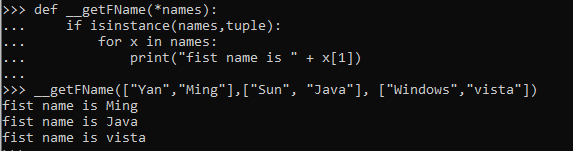
##### Creation:

Please refer to my previous examples.

##### Arbitrary arguments, \*args

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.

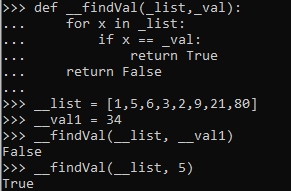
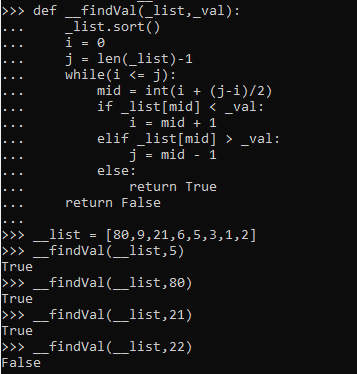
This way the function will receive a **tuple of arguments**, and can access the items accordingly:



##### Keyword Arguments

You can also send arguments with the key = value syntax. This way the order of the arguments does not matter.

Without keyword:

 Or 

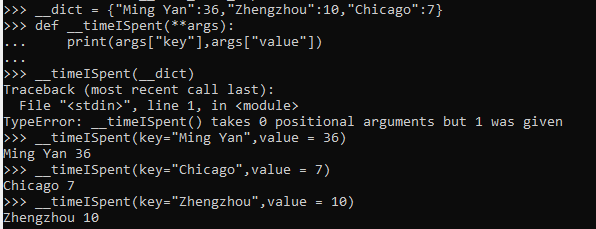
With keyword:



##### Arbitrary Keyword Arguments, \*\*kwargs

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: \*\* before the parameter name in the function definition.

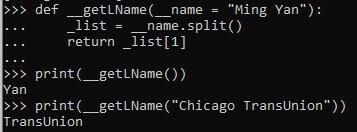
This way the function will receive a **dictionary of arguments**, and can access the items accordingly:



##### Default Parameter Value

The following example shows how to use a default parameter value.

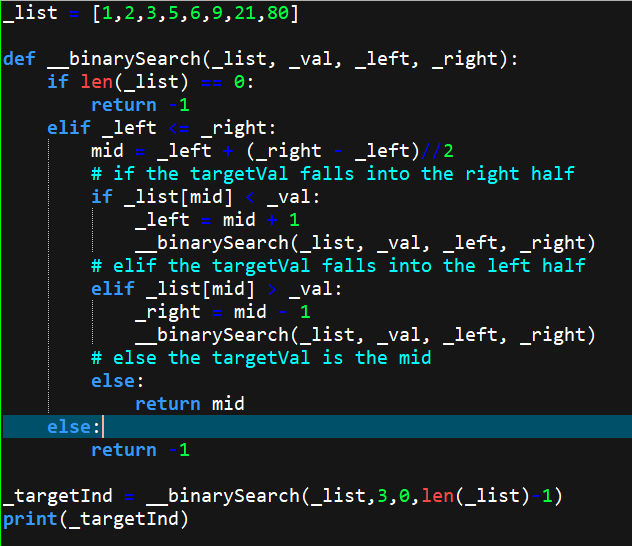
If we call the function without argument, it uses the default value:



##### The pass statement

Function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error. Same as if statement.

##### Recursion



#### Lambda

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Java also has this, only we call it Lambda expression, instead of function even it works as one.

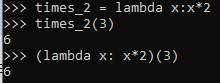
##### Syntax

lambda arguments : expression

The definition of the lambda lists the arguments with no parentheses, whereas calling the function is done exactly like a normal Python function, with parentheses surrounding the arguments.



You can apply the function above to an argument by surrounding the function and its argument with parentheses:



This starts looking like an anonymous function in Java using lambda expression.

What happened is actualy

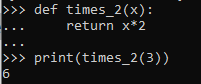
(lambda x : x\*2)(3) = lambda 3: 3\*2

= 3\*2

= 6

Because a lambda function is an expression, (yeah, I still prefer to call it an expression). It can be named. So that’s what the times\_2 is for.

And it is equivalent to:



Lambda functions can take any number of arguments



and



Why do we need lambda functions?

In Python, an anonymous function is created with the lambda keyword. More loosely, it may or not be assigned a name.

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

Pending, may not be required for xgboost

### Iterator

### Classes and Objects

Same as Java, blah blah blah

For more information, refer to [API](https://docs.python.org/3/tutorial/classes.html)

Compared with other programming languages, Python’s class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in **C++ and Modula-3**. Python classes provide all the standard features of Object Oriented Programming: the class inheritance mechanism allows multiple base classes, a derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation.

Class definitions play some neat tricks with namespaces, and you need to know how scopes and namespaces work to fully understand what’s going on. **Incidentally, knowledge about this subject is useful for any advanced Python programmer.**

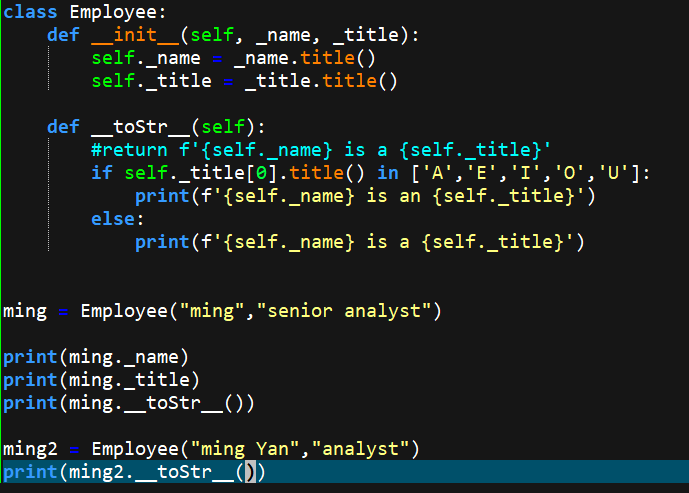
#### Create a class

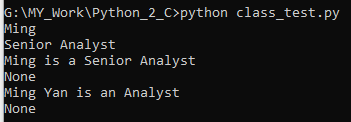
Same as Java

#### The \_\_init\_\_() Function

All classes have a function called \_\_init\_\_(), **which is always executed when the class is being initiated.**

Use the \_\_init\_\_() function to assign values to object properties, or other operations that are necessary to do when the object is being created:





### Inheritance

### Modules

#### json

#### pickle

Refer to [API](https://docs.python.org/3/library/pickle.html#module-pickle)

dump(), load()

#### xgboost

refer to [API](https://xgboost.readthedocs.io/en/latest/python/python_api.html)

>>> import pickle

>>> with open("prm.edtin.dgmalcu1.input.20211105\_120215\_50124956","rb") as f:

... file = pickle.load(f)

...

>>> type(file)

<class 'dict'>

>>> file.keys()

dict\_keys(['Model', 'Partial\_Dependencies'])

>>> model = file["Model"]

>>> type(model)

<class 'xgboost.core.Booster'>

>>> import xgboost as xgb

>>> isinstance(model, xgb.Booster)

True

>>> model.booster

'gbtree'

>>> config = model.save\_config()

>>> import json

>>> parameter = json.loads(config)

>>> parameter

{'learner': {'generic\_param': {'fail\_on\_invalid\_gpu\_id': '0', 'gpu\_id': '-1', 'gpu\_page\_size': '0', 'n\_gpus': '0', 'n\_jobs': '8', 'nthread': '8', 'random\_state': '1441', 'seed': '1441', 'seed\_per\_iteration': '0', 'validate\_parameters': '1'}, 'gradient\_booster': {'gbtree\_train\_param': {'num\_parallel\_tree': '1', 'predictor': 'auto', 'process\_type': 'default', 'tree\_method': 'exact', 'updater': 'grow\_colmaker,prune', 'updater\_seq': 'grow\_colmaker,prune'}, 'name': 'gbtree', 'specified\_updater': False, 'updater': {'grow\_colmaker': {'colmaker\_train\_param': {'opt\_dense\_col': '1'}, 'train\_param': {'alpha': '10', 'cache\_opt': '1', 'colsample\_bylevel': '0.600000024', 'colsample\_bynode': '1', 'colsample\_bytree': '0.699999988', 'default\_direction': 'learn', 'enable\_feature\_grouping': '0', 'eta': '0.00100000005', 'gamma': '1', 'grow\_policy': 'depthwise', 'interaction\_constraints': '', 'lambda': '300', 'learning\_rate': '0.00100000005', 'max\_bin': '256', 'max\_conflict\_rate': '0', 'max\_delta\_step': '0', 'max\_depth': '6', 'max\_leaves': '0', 'max\_search\_group': '100', 'min\_child\_weight': '120', 'min\_split\_loss': '1', 'monotone\_constraints': '(-1,-1,-1,-1,-1,1,1,1,1,1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1)', 'refresh\_leaf': '1', 'reg\_alpha': '10', 'reg\_lambda': '300', 'sampling\_method': 'uniform', 'sketch\_eps': '0.0299999993', 'sketch\_ratio': '2', 'sparse\_threshold': '0.20000000000000001', 'split\_evaluator': 'elastic\_net,monotonic', 'subsample': '0.632000029'}}, 'prune': {'train\_param': {'alpha': '10', 'cache\_opt': '1', 'colsample\_bylevel': '0.600000024', 'colsample\_bynode': '1', 'colsample\_bytree': '0.699999988', 'default\_direction': 'learn', 'enable\_feature\_grouping': '0', 'eta': '0.00100000005', 'gamma': '1', 'grow\_policy': 'depthwise', 'interaction\_constraints': '', 'lambda': '300', 'learning\_rate': '0.00100000005', 'max\_bin': '256', 'max\_conflict\_rate': '0', 'max\_delta\_step': '0', 'max\_depth': '6', 'max\_leaves': '0', 'max\_search\_group': '100', 'min\_child\_weight': '120', 'min\_split\_loss': '1', 'monotone\_constraints': '(-1,-1,-1,-1,-1,1,1,1,1,1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1)', 'refresh\_leaf': '1', 'reg\_alpha': '10', 'reg\_lambda': '300', 'sampling\_method': 'uniform', 'sketch\_eps': '0.0299999993', 'sketch\_ratio': '2', 'sparse\_threshold': '0.20000000000000001', 'split\_evaluator': 'elastic\_net,monotonic', 'subsample': '0.632000029'}}}}, 'learner\_model\_param': {'base\_score': '5E-1', 'num\_class': '0', 'num\_feature': '100'}, 'learner\_train\_param': {'booster': 'gbtree', 'disable\_default\_eval\_metric': '0', 'dsplit': 'auto', 'objective': 'binary:logistic'}, 'metrics': ['aucpr'], 'objective': {'name': 'binary:logistic', 'reg\_loss\_param': {'scale\_pos\_weight': '1'}}}, 'version': [1, 4, 2]}

>>> type(parameter)

<class 'dict'>

>>> parameter.keys()

dict\_keys(['learner', 'version'])

>>> parameter.keys()

dict\_keys(['learner', 'version'])

>>> parameter["learner"]

{'generic\_param': {'fail\_on\_invalid\_gpu\_id': '0', 'gpu\_id': '-1', 'gpu\_page\_size': '0', 'n\_gpus': '0', 'n\_jobs': '8', 'nthread': '8', 'random\_state': '1441', 'seed': '1441', 'seed\_per\_iteration': '0', 'validate\_parameters': '1'}, 'gradient\_booster': {'gbtree\_train\_param': {'num\_parallel\_tree': '1', 'predictor': 'auto', 'process\_type': 'default', 'tree\_method': 'exact', 'updater': 'grow\_colmaker,prune', 'updater\_seq': 'grow\_colmaker,prune'}, 'name': 'gbtree', 'specified\_updater': False, 'updater': {'grow\_colmaker': {'colmaker\_train\_param': {'opt\_dense\_col': '1'}, 'train\_param': {'alpha': '10', 'cache\_opt': '1', 'colsample\_bylevel': '0.600000024', 'colsample\_bynode': '1', 'colsample\_bytree': '0.699999988', 'default\_direction': 'learn', 'enable\_feature\_grouping': '0', 'eta': '0.00100000005', 'gamma': '1', 'grow\_policy': 'depthwise', 'interaction\_constraints': '', 'lambda': '300', 'learning\_rate': '0.00100000005', 'max\_bin': '256', 'max\_conflict\_rate': '0', 'max\_delta\_step': '0', 'max\_depth': '6', 'max\_leaves': '0', 'max\_search\_group': '100', 'min\_child\_weight': '120', 'min\_split\_loss': '1', 'monotone\_constraints': '(-1,-1,-1,-1,-1,1,1,1,1,1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1)', 'refresh\_leaf': '1', 'reg\_alpha': '10', 'reg\_lambda': '300', 'sampling\_method': 'uniform', 'sketch\_eps': '0.0299999993', 'sketch\_ratio': '2', 'sparse\_threshold': '0.20000000000000001', 'split\_evaluator': 'elastic\_net,monotonic', 'subsample': '0.632000029'}}, 'prune': {'train\_param': {'alpha': '10', 'cache\_opt': '1', 'colsample\_bylevel': '0.600000024', 'colsample\_bynode': '1', 'colsample\_bytree': '0.699999988', 'default\_direction': 'learn', 'enable\_feature\_grouping': '0', 'eta': '0.00100000005', 'gamma': '1', 'grow\_policy': 'depthwise', 'interaction\_constraints': '', 'lambda': '300', 'learning\_rate': '0.00100000005', 'max\_bin': '256', 'max\_conflict\_rate': '0', 'max\_delta\_step': '0', 'max\_depth': '6', 'max\_leaves': '0', 'max\_search\_group': '100', 'min\_child\_weight': '120', 'min\_split\_loss': '1', 'monotone\_constraints': '(-1,-1,-1,-1,-1,1,1,1,1,1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1)', 'refresh\_leaf': '1', 'reg\_alpha': '10', 'reg\_lambda': '300', 'sampling\_method': 'uniform', 'sketch\_eps': '0.0299999993', 'sketch\_ratio': '2', 'sparse\_threshold': '0.20000000000000001', 'split\_evaluator': 'elastic\_net,monotonic', 'subsample': '0.632000029'}}}}, 'learner\_model\_param': {'base\_score': '5E-1', 'num\_class': '0', 'num\_feature': '100'}, 'learner\_train\_param': {'booster': 'gbtree', 'disable\_default\_eval\_metric': '0', 'dsplit': 'auto', 'objective': 'binary:logistic'}, 'metrics': ['aucpr'], 'objective': {'name': 'binary:logistic', 'reg\_loss\_param': {'scale\_pos\_weight': '1'}}}

>>> type(parameter["learner"])

<class 'dict'>

>>> parameter["learner"].keys()

dict\_keys(['generic\_param', 'gradient\_booster', 'learner\_model\_param', 'learner\_train\_param', 'metrics', 'objective'])

>>> parameter["learner"]["learner\_model\_param"]

{'base\_score': '5E-1', 'num\_class': '0', 'num\_feature': '100'}

>>> type(parameter["learner"]["learner\_model\_param"])

<class 'dict'>

>>> parameter["learner"]["objective"]

{'name': 'binary:logistic', 'reg\_loss\_param': {'scale\_pos\_weight': '1'}}

>>> parameter["learner"]["generic\_param"]

{'fail\_on\_invalid\_gpu\_id': '0', 'gpu\_id': '-1', 'gpu\_page\_size': '0', 'n\_gpus': '0', 'n\_jobs': '8', 'nthread': '8', 'random\_state': '1441', 'seed': '1441', 'seed\_per\_iteration': '0', 'validate\_parameters': '1'}

>>> parameter["learner"]["gradient\_booster"]

{'gbtree\_train\_param': {'num\_parallel\_tree': '1', 'predictor': 'auto', 'process\_type': 'default', 'tree\_method': 'exact', 'updater': 'grow\_colmaker,prune', 'updater\_seq': 'grow\_colmaker,prune'}, 'name': 'gbtree', 'specified\_updater': False, 'updater': {'grow\_colmaker': {'colmaker\_train\_param': {'opt\_dense\_col': '1'}, 'train\_param': {'alpha': '10', 'cache\_opt': '1', 'colsample\_bylevel': '0.600000024', 'colsample\_bynode': '1', 'colsample\_bytree': '0.699999988', 'default\_direction': 'learn', 'enable\_feature\_grouping': '0', 'eta': '0.00100000005', 'gamma': '1', 'grow\_policy': 'depthwise', 'interaction\_constraints': '', 'lambda': '300', 'learning\_rate': '0.00100000005', 'max\_bin': '256', 'max\_conflict\_rate': '0', 'max\_delta\_step': '0', 'max\_depth': '6', 'max\_leaves': '0', 'max\_search\_group': '100', 'min\_child\_weight': '120', 'min\_split\_loss': '1', 'monotone\_constraints': '(-1,-1,-1,-1,-1,1,1,1,1,1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1)', 'refresh\_leaf': '1', 'reg\_alpha': '10', 'reg\_lambda': '300', 'sampling\_method': 'uniform', 'sketch\_eps': '0.0299999993', 'sketch\_ratio': '2', 'sparse\_threshold': '0.20000000000000001', 'split\_evaluator': 'elastic\_net,monotonic', 'subsample': '0.632000029'}}, 'prune': {'train\_param': {'alpha': '10', 'cache\_opt': '1', 'colsample\_bylevel': '0.600000024', 'colsample\_bynode': '1', 'colsample\_bytree': '0.699999988', 'default\_direction': 'learn', 'enable\_feature\_grouping': '0', 'eta': '0.00100000005', 'gamma': '1', 'grow\_policy': 'depthwise', 'interaction\_constraints': '', 'lambda': '300', 'learning\_rate': '0.00100000005', 'max\_bin': '256', 'max\_conflict\_rate': '0', 'max\_delta\_step': '0', 'max\_depth': '6', 'max\_leaves': '0', 'max\_search\_group': '100', 'min\_child\_weight': '120', 'min\_split\_loss': '1', 'monotone\_constraints': '(-1,-1,-1,-1,-1,1,1,1,1,1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,1,-1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1)', 'refresh\_leaf': '1', 'reg\_alpha': '10', 'reg\_lambda': '300', 'sampling\_method': 'uniform', 'sketch\_eps': '0.0299999993', 'sketch\_ratio': '2', 'sparse\_threshold': '0.20000000000000001', 'split\_evaluator': 'elastic\_net,monotonic', 'subsample': '0.632000029'}}}}

>>> parameter["learner"]["learner\_model\_param"]

{'base\_score': '5E-1', 'num\_class': '0', 'num\_feature': '100'}

>>> parameter["learner"]["learner\_model\_param"]

{'base\_score': '5E-1', 'num\_class': '0', 'num\_feature': '100'}

>>> parameter["learner"]["gradient\_booster"].keys()

dict\_keys(['gbtree\_train\_param', 'name', 'specified\_updater', 'updater'])

>>> parameter["learner"]["gradient\_booster"]["gbtree\_train\_param"].keys()

dict\_keys(['num\_parallel\_tree', 'predictor', 'process\_type', 'tree\_method', 'updater', 'updater\_seq'])

>>> parameter["learner"]["gradient\_booster"]["gbtree\_train\_param"]

{'num\_parallel\_tree': '1', 'predictor': 'auto', 'process\_type': 'default', 'tree\_method': 'exact', 'updater': 'grow\_colmaker,prune', 'updater\_seq': 'grow\_colmaker,prune'}

# dump the model to a list of string

>>> model\_dump = model.get\_dump()

>>> type(model\_dump)

<class 'list'>

>>> model\_dump[0]

'0:[eads52prrev205<4.94499969] yes=1,no=2,missing=2\n\t1:[eads05agg911<105.785004] yes=3,no=4,missing=4\n\t\t3:[eads142at36s<0.5] yes=5,no=6,missing=6\n\t\t\t5:leaf=-0.000246113283\n\t\t\t6:leaf=-0.000712651934\n\t\t4:leaf=0.000113421556\n\t2:leaf=-0.00147898577\n'

>>> type(model\_dump[0])

<class 'str'>

>>> model\_dump = model.get\_dump(dump\_format='json')

>>> type(model\_dump)

<class 'list'>

>>> type(model\_dump[0])

<class 'str'>

>>> model\_dump\_tree\_0 = json.loads(model\_dump[0])

>>> type(model\_dump\_tree\_0)

<class 'dict'>

>>> model\_dump\_tree\_0.keys()

dict\_keys(['nodeid', 'depth', 'split', 'split\_condition', 'yes', 'no', 'missing', 'children'])

>>> for key,value in model\_dump\_tree\_0.items():

... print(key,":",value)

...

nodeid : 0

depth : 0

split : eads52prrev205

split\_condition : 4.94499969

yes : 1

no : 2

missing : 2

children : [{'nodeid': 1, 'depth': 1, 'split': 'eads05agg911', 'split\_condition': 105.785004, 'yes': 3, 'no': 4, 'missing': 4, 'children': [{'nodeid': 3, 'depth': 2, 'split': 'eads142at36s', 'split\_condition': 0.5, 'yes': 5, 'no': 6, 'missing': 6, 'children': [{'nodeid': 5, 'leaf': -0.000246113283}, {'nodeid': 6, 'leaf': -0.000712651934}]}, {'nodeid': 4, 'leaf': 0.000113421556}]}, {'nodeid': 2, 'leaf': -0.00147898577}]

# get the attribute list, maybe can be used in the feature.csv

>>> model\_fscore = model.get\_fscore()

>>> type(model\_fscore)

<class 'dict'>

>>> model\_fscore.keys()

dict\_keys(['eads52prrev205', 'eads05agg911', 'eads142at36s', 'eads142re102s', 'eads52prrev225', 'eads05agg905', 'eads05agg114', 'eads06trv01', 'eads142bc102s', 'eads142g244b', 'eads52epall231', 'eads52epall254', 'eads142s114s', 'eads142br21s', 'eads142g213a', 'eads142re36s', 'eads142in34s', 'eads142at31s', 'eads52obbkc324', 'eads142g990s', 'eads142g417s', 'eads142at34b', 'eads52prrev204', 'eads15indexq2', 'eads142at21s', 'eads11cv21', 'eads05agg108', 'eads15waggs909', 'eads15rvrvdex02', 'eads15aggs114', 'eads15walshrs2', 'eads05agg401', 'eads142s004s', 'eads142br20s', 'eads11cv26', 'eads05agg205', 'eads15rvrevs123', 'eads142bc21s', 'eads142s064a', 'eads142au51a', 'eads142at28a', 'eads05agg110', 'eads15rvrvdexq2', 'eads52prrev201', 'eads08rvlr29', 'eads15waggs908', 'eads52obrev321', 'eads52prrev202', 'eads05agg519', 'eads11cv19', 'eads15indexq1', 'eads15rvrevs108', 'eads15rvrevs904', 'eads142at101s', 'eads142bc20s', 'eads142g205s', 'eads142at104s', 'eads05agg116', 'eads11cv28', 'eads15rvwalsrvs2', 'eads142at35b', 'eads05agg106', 'eads15rvrevs105', 'eads05agg119', 'eads142at20s', 'eads15rtrtdex01', 'eads05agg404', 'eads142at101b', 'eads142g102s', 'eads05agg909', 'eads05agg423', 'eads15rvrvdexq3', 'eads05agg101', 'eads52epaut235', 'eads05agg402', 'eads05agg419', 'eads142g199s', 'eads142bc103s', 'eads142at01s', 'eads142in21s', 'eads15rtrets124', 'eads142g207s', 'eads52uzbkc83', 'eads10paymnt07', 'eads05agg420', 'eads142re20s', 'eads142au20s', 'eads142rt20s', 'eads142re101s', 'eads142in20s', 'eads142fi20s', 'eads15rvrvdexq1', 'eads142of20s', 'eads142reap01', 'eads07balmag01', 'eads11cv27', 'eads142rt21s', 'eads07balmag02', 'eads142bc97b', 'eads142g106s'])

>>> for attrName, importance in model\_fscore.items():

... print(attrName, ":", importance)

...

eads52prrev205 : 2008

eads05agg911 : 6733

eads142at36s : 3751

eads142re102s : 1549

eads52prrev225 : 2370

eads05agg905 : 4220

eads05agg114 : 404

eads06trv01 : 4020

eads142bc102s : 1529

eads142g244b : 720

eads52epall231 : 1717

eads52epall254 : 524

eads142s114s : 2345

eads142br21s : 639

eads142g213a : 3377

eads142re36s : 2611

eads142in34s : 702

eads142at31s : 3507

eads52obbkc324 : 7379

eads142g990s : 1613

eads142g417s : 1125

eads142at34b : 4062

eads52prrev204 : 927

eads15indexq2 : 1813

eads142at21s : 2262

eads11cv21 : 1006

eads05agg108 : 840

eads15waggs909 : 679

eads15rvrvdex02 : 1203

eads15aggs114 : 995

eads15walshrs2 : 3659

eads05agg401 : 419

eads142s004s : 1334

eads142br20s : 308

eads11cv26 : 483

eads05agg205 : 265

eads15rvrevs123 : 1800

eads142bc21s : 2322

eads142s064a : 1413

eads142au51a : 1161

eads142at28a : 2937

eads05agg110 : 529

eads15rvrvdexq2 : 1769

eads52prrev201 : 776

eads08rvlr29 : 226

eads15waggs908 : 719

eads52obrev321 : 7934

eads52prrev202 : 1460

eads05agg519 : 774

eads11cv19 : 1281

eads15indexq1 : 2153

eads15rvrevs108 : 362

eads15rvrevs904 : 28

eads142at101s : 767

eads142bc20s : 304

eads142g205s : 119

eads142at104s : 2206

eads05agg116 : 719

eads11cv28 : 334

eads15rvwalsrvs2 : 1044

eads142at35b : 132

eads05agg106 : 786

eads15rvrevs105 : 1032

eads05agg119 : 637

eads142at20s : 36

eads15rtrtdex01 : 2105

eads05agg404 : 563

eads142at101b : 263

eads142g102s : 1343

eads05agg909 : 2367

eads05agg423 : 115

eads15rvrvdexq3 : 1502

eads05agg101 : 76

eads52epaut235 : 711

eads05agg402 : 121

eads05agg419 : 950

eads142g199s : 732

eads142bc103s : 125

eads142at01s : 1115

eads142in21s : 2052

eads15rtrets124 : 1498

eads142g207s : 718

eads52uzbkc83 : 2943

eads10paymnt07 : 77

eads05agg420 : 50

eads142re20s : 110

eads142au20s : 919

eads142rt20s : 136

eads142re101s : 18

eads142in20s : 2012

eads142fi20s : 4361

eads15rvrvdexq1 : 257

eads142of20s : 1200

eads142reap01 : 19

eads07balmag01 : 55

eads11cv27 : 49

eads142rt21s : 7

eads07balmag02 : 333

eads142bc97b : 12

eads142g106s : 15

**maybe we could figure out the order ourselves.**

>>> parameter["learner"].keys()

dict\_keys(['generic\_param', 'gradient\_booster', 'learner\_model\_param', 'learner\_train\_param', 'metrics', 'objective'])

>>> parameter["learner"]["generic\_param"]

{'fail\_on\_invalid\_gpu\_id': '0', 'gpu\_id': '-1', 'gpu\_page\_size': '0', 'n\_gpus': '0', 'n\_jobs': '8', 'nthread': '8', 'random\_state': '1441', 'seed': '1441', 'seed\_per\_iteration': '0', 'validate\_parameters': '1'}

>>> parameter["learner"]["generic\_param"].keys()

dict\_keys(['fail\_on\_invalid\_gpu\_id', 'gpu\_id', 'gpu\_page\_size', 'n\_gpus', 'n\_jobs', 'nthread', 'random\_state', 'seed', 'seed\_per\_iteration', 'validate\_parameters'])

>>> parameter["learner"]["gradient\_booster"].keys()

dict\_keys(['gbtree\_train\_param', 'name', 'specified\_updater', 'updater'])

>>> parameter["learner"]["gradient\_booster"]["gbtee\_train\_param"]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

KeyError: 'gbtee\_train\_param'

>>> parameter["learner"]["gradient\_booster"]["gbtree\_train\_param"]

{'num\_parallel\_tree': '1', 'predictor': 'auto', 'process\_type': 'default', 'tree\_method': 'exact', 'updater': 'grow\_colmaker,prune', 'updater\_seq': 'grow\_colmaker,prune'}