GAME CENTER

BURKINA 74 91 21 62

omastarliliou@gmail.com

CH2 documentation

WHAT'S NEW?

- With ch2, there is no more space constraint. As long as you have a colon (:), anything on the left will be considered the key, and anything on the right will be considered the value.

- ch2 is typed, after retrieving the value, you can know the type which is last_data_type.

- ch2 can contain a table. Once imported, it will be a list.

- ch2 also provides an object, ch_file, which allows you to use ch2 in RAM to increase the speed of your programs.

- ch2 provides a verifiertype function returning a string corresponding to the type of the input value.

- ch_data is no longer case sensitive

- ch_getAll retrieves the entire ch file into a dictionary. By itself it is enough to handle all ch.

- There is also GCS_interpreter to allow you to run your own gcs script cores. Here is an example of how to use it.

- ch2 provides the Vars object that can be used with GCS_interpreter.

PART 1: CH2

ch2 is used to store data used in your program. It includes data and sections. By convention, a ch file starts with a comment with its name and an #END comment to indicate the end. But it is not mandatory.

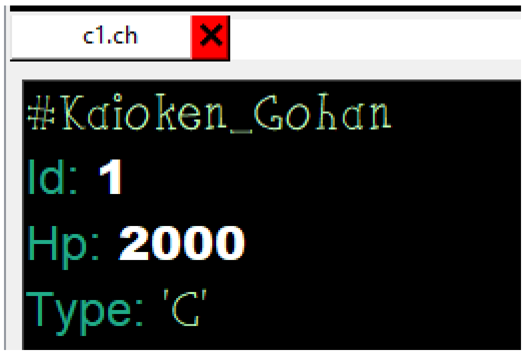
1) Data.

we write data to a ch file in this way:

key : value

key2 : value

* key and value can now be written any way. Ch2 has no space constraints, so there is no more normalize! parameter.



```
#Kaioken_Gohan
Id: 1
Hp: 2000
Type: 'C'
```

A ch file containing the data of the characters of a game

2) The comments

Just like in python, you can comment a ch line with #. If what you want is a multi-line comment, then just create a section.

Plus, if you put an #END comment in your ch file, anything you add below it will be completely ignored. So it's a good place for multi-line comments, or even to tell your life story!

3) Sections.

We write the ch sections this way:

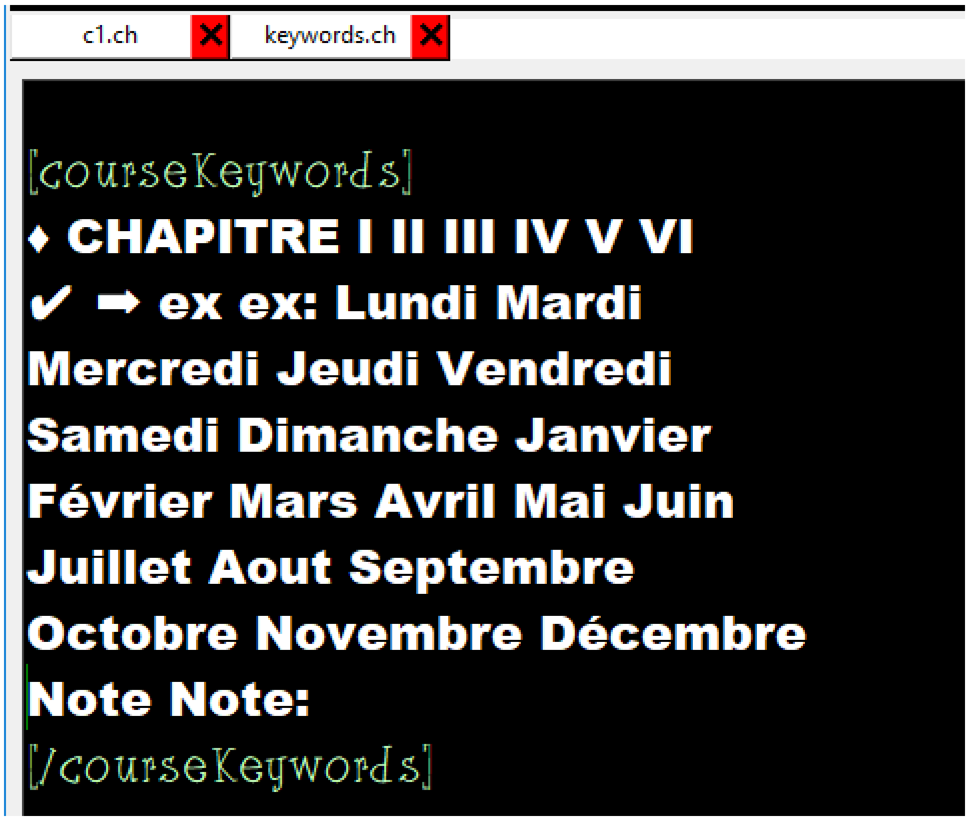
```
[section_name]

Text in the section

Several lines without

problems[/section_name]
```

write the section without worrying about spaces, ch will not change a letter!



c1.ch keywords.ch

```
[courseKeywords]
♦ CHAPITRE I II III IV V VI
✔ ➡ ex ex: Lundi Mardi
Mercredi Jeudi Vendredi
Samedi Dimanche Janvier
Février Mars Avril Mai Juin
Juillet Aout Septembre
Octobre Novembre Décembre
Note Note:
[/courseKeywords]
```

A ch section containing keywords

PART 2: UTILITY

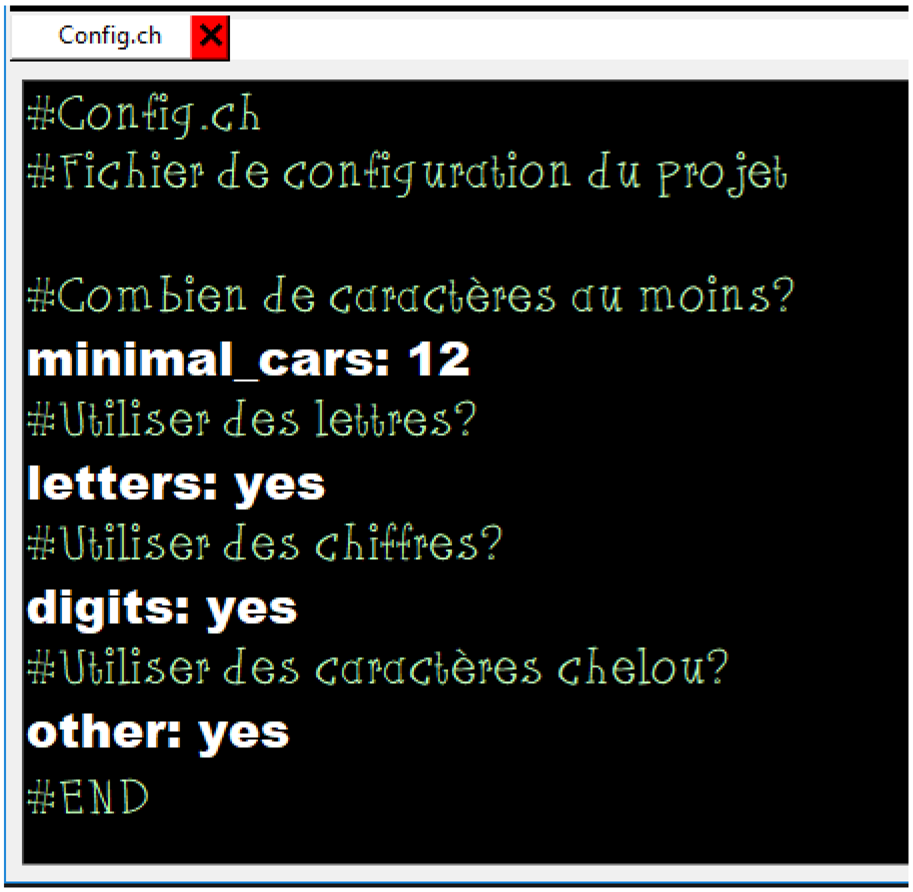
What is it for?

Setup:

To set up your applications. You can save your settings to retrieve them later.

Used in parameterization, the file is not modified by the program. It is a good use to create multilingual programs for example.

Your program's values can then be filled in by users. The parameter reading functions are ch_data, ch_section and ch_data_section.


Config.ch

```
#Config.ch
#Fichier de configuration du projet

#Combien de caractères au moins?
minimal_cars: 12
#Utiliser des lettres?
letters: yes
#Utiliser des chiffres?
digits: yes
#Utiliser des caractères chelou?
other: yes
#END
```

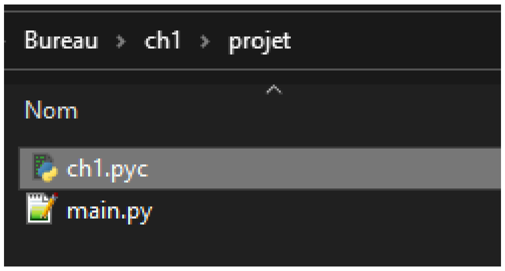
A Configuration file ch. Allows you to change the behavior of the of the program.

Dynamic:

The program saves and reads the data it needs by itself without anyone touching it. This is not often used for setting parameters but rather for saving. The dynamic functions are ch_update, ch_create, ch_drop.

PART 3: HOW TO USE IT?

1) put the ch2.pyc file in your project folder. If you want to install it permanently (and thus avoid copying it in all your projects), paste it in the Lib folder of your python.
(Search for python, then go to the location ... twice!)



ch1.pyc in your project folder

2) import the modules of your choice

ex: from ch2 import *

3) use the function of your choice

ex: ch_create("test")

The functions of ch2:

open a python project, and try it as you go. Are you ready?

1) ch_create(file = "configurations")

ch_create only creates a ch file. It is not necessary to specify the file extension. If no parameter is specified, the file created will be configurations.ch

example: ch_create("test.ch") #creates a file test.ch

2) ch_data(data, file, strip = True)

ch_data returns the value of a ch key. By default, it "strips" the value, i.e. it removes spaces it deems unnecessary.

example:

ch_data("cle1", "test.ch") #returns the value of cle1 in the file test.ch ch_data("cle1:",

"test.ch") #The ch1 syntax works too!

#in the ch:

cle1:value1 #returns "value 1" by default,   "value1" if strip is False.

3) ch_update(data, file)

ch_update modifies a ch file. This time, data contains not only the key, but the whole line.

ch_update("key1: new_value", "test.ch") #If key1 exists in the file, it is updated. Otherwise, it is just added at the end.

4) ch_section(section, file)

ch_section retrieves a section rather than a value. The section is not changed in any way.

ch_section("big_text", "test.ch")

#in the ch file:

[big_text]

this is a huge text

stored in file

[/big_text]

5) ch_drop(data, file)

ch_drop removes the line containing data

ex: ch_drop("cle1:", "test.ch") #deletes cle1 from the ch file

6) ch_section_data(data, section, file)

ch_section_data returns ch data in a specific section.

ex: ch_section_data("cle1:", "section1", "test.ch") #returns the key1 in section1, ignores if it exists in other sections.

7) ch_getAll(file, strip = True)

ch_getAll retrieves all registered ch keys and values, as well as all sections, into a dictionary. Since the return value is a dictionary, the data becomes case sensitive again.

ex: my_dict = ch_getAll() #return { key1 : value1, key2 : value2 }

8) verifiertype(char)

verifiertype returns "int", "float", "string" or "none" depending on the type of the parameter.

9) last_data_type

This is a variable that will change a lot during the program. It will allow you to know the type of the last value retrieved from a ch file.

10) ch_file

ch_file is a class to use ch virtually. It is loaded at the beginning of the program, and saved if needed at the end.
ch_file has the following methods:

* ch_data(self, key, strip=True):
* ch_section(self, section):
* ch_getAll(self, strip=True):
* ch_update(self, data, value=""):
* addFile(self, filename):
* save(self):
* setCurrentFile(self, filename):
* ch_section_data(self, data, section):
* ch_drop(self, data):
* printCurrentFile(self):

PART 4: EXAMPLES

Multilingual program

#This is the main.py file, containing a program that is supposed to say "Salut, je parle Francais!" or "Hello, i speak English!" depending on the current language.

```
#We import ch_data

from ch2 import ch_data

#The languages to be used are

defined fr = "fr.ch".

eng = "eng.ch

#The current

language

actual_lang = en


#Display in current language

print(ch_data("hi" , actual_lang))
```

In the same folder, we will then have the fr.ch and eng.ch files that we declared above! Here are their respective contents :

```
#en.ch

hi: Hi, I speak French!


#eng.ch

hi : hello! i speak English!
```

You can continue with this logic and add as many languages as you want, keeping the keys the same as "hi" here!

Change the language?

I'm sure you can do it yourself. Just in case...

```
print(ch_data( "thing:" , actual_lang))

choice = int(input())

if(choice == 1):
        actual_lang = eng
else:
        actual_lang = en
```

With in the files fr.ch and eng.ch :

```
#en.ch
hi: Hi, I speak French!
choose: Please type 1 to use English and something else to stay in French.


#eng.ch
hi : hello! ,i speak English!
choose: Please press 1 if you want to use english, anything else if you want to use french.
```

Well, there you go. You already knew that, right? This is just a small example, but imagine a complete program coded in multi-languages. What is also nice is that you can modify your ch file afterwards without touching your code!

Program recording its data

Let's go into the context of an equally simple application. A program that needs for example to retain a fixed value. Let's say a game for example that keeps the best recorded score.

```
#File params.ch
best_score : 100


#file main.py
print( "The best score is " +str(ch_data( "best_score" , "params.ch" ))) #Attention : ch2 is typed, it returns 100 in number, not in string!
```

That best score there, quit your program and run it again; it will still be the same. Normal, it is written in a file. So, a player plays our game and at the end he has a score. How do we check if it's the best score, now?

```
#It will be necessary to include ch_update above

bestScore = ch_data( "best_score:" , "params.ch" )

if(scorePlayer>bestScore):

    ch_update( "best_score: " +str(scorePlayer), "params.ch" )
```

Well, there you go. There are many other ways to use ch, it's up to you!

Program reading data, but using ch_file

This time we'll make a fast program that will make as few transactions as possible to the ch file. For that, let's make a program with two files : config.ch and authorizations.ch

```
#In main.py from

ch2 import *


#First, create the object db

= ch_file(''config.ch'')

db.addFile(''authorisations.ch'')



materials_views = db.ch_data(''materials'')
```

```
number_of_students =

db.ch_data("number_of_students")

db.setCurrentFile("authorisations.ch")

permission_read = db.ch_data("read")

permission_write = db.ch_data("write")

permission_super = db.ch_data("all")


#We can test everything!

print ("In CEP1, we see the following subjects:")

for matiere in matieres_vues :

            print(material)

print ( "The number of students is :

"+str(number_students) ) if permission_read ==

'yes' :

            print( "This program can read! " ) if

permission_writing == 'yes' :

            print( "This program can write! " ) if

permission_super == 'yes' :

            print( "This do-it-all program! " )


#config.ch

subjects : ( Dictation, Calculus,

Copy ) number of students : 23

#END


#authorizations.c

h read: yes
```

write: yes

all: yes

```
#If you modify the file, don't forget to save it. #main.py

db.ch_update( 'all', 'no')

db.save()
```

USE OF GCS INTERPRETER

```
#We need GCS_interpreter from
ch2 import *

#We create our heir class
myCore(GCS_interpreter):
  def init (self):
    super(). init ()

        #We add commands.
        #The command is the first word of a gcs line
        #All words following the command will be sent to the function as a list
    self.add_command("print", self.affiche)

        #params is the collected list (everything after the command in the gcs file) #Write the
        function of your choice.
        #You must always take a list parameter.
        #All the values in the list are of type string, so we will have to convert them.
  def affiche(self, params=[]):
    print(' '.join(params))

myCore().run("test.gcs")


#In the file test.gcs :
```

```
print hello
world! end
```

Vars is also included. This will allow you to use it in your GCS core. example :

```
#We need GCS_interpreter from
ch2 import *

#We create our heir class
myCore(GCS_interpreter):

   def init (self):
      super(). init ()
           self.vars = Vars()
           self.vars.setvar("*UN", 1)
           self.vars.setvar("*TWO", 2)

           #We add commands.
           #The command is the first word of a gcs line
           #All words following the command will be sent to the function as a list
      self.add_command("addition", self.add)

#params is the collected list (everything after the command in the gcs file) #Write the
function of your choice.
#You must always take a list parameter.
#All the values in the list are of type string, so they must be converted.

def add(self, params=[]):
      print(self.vars.getvar("*UN") + self.vars.getvar("*TWO"))
```

Class Description:

```
class Var:
           def getvar(self, var= "LASTRESULT" )
```

```
        def contains(self, var="LASTRESULT")
        def setvar(self, var, value= "" )
        def remvar(self,
        var) def
        remvars(self)
        def savevars(self, savefile = "variables.ch"
        ) def getvars(self, savefile = "variables.ch"
        ) def printvars(self)

class GCS_interpreter:
        def add_command(self, command, function) #Command name, command function
        def run(self, filename) #Filename, or lines list from ch_section.split('\n')
        def error(self, no) #Error no
```

Download ch2 on our Game Center

facebook page