SORAY CENGİZ
161024077

HW04:

13.1.1 pickle-storage-loading-python

13.1.1.1 pickle_file_manip.py

Python pickle modülü, bir Python nesne yapısını serileştirmek ve serileştirmeyi kaldırmak için kullanılır. Python'daki herhangi nesne diske kaydedilebilir. Pickle bir python nesnesini (list, dict, etc.) bir karakter akışına dönüştürmenin bir yoludur. Buradaki fikir, bu karakter akışının, nesneyi başka bir python script'inde yeniden yapılandırmak için gerekli tüm bilgiyi içermesidir.

Pickle kullanarak nesneyi binary formatta bir dosyaya kaydediyoruz, daha sonra bu dosya kullanılarak başka bir script'de kaydedilen nesne yeniden yapılandırılabilir. Herhangi bir veri bütününü kaydetmek isteseydik python'da birkaç farklı modül kullanabilirdik numpy ile basit dosya kayıt işlemleri yapılabilir, "json" modülü ile yine birbiri ile ilişkili veriler dosyaya kaydedilebilir, "csv" modülü ile yine aynı şekilde binary olmayan dosyalara veriler kaydedilebilir veya veriler çekilebilir.

In Python:

```
1  #!/usr/bin/python
2
3  # /*
4  #  * This file is part of the "dev-in-place" repository located at:
5  #  * https://github.com/osuvak/dev-in-place
6  #  *
7  #  * Copyright (C) 2019  Onder Suvak
8  #  *
9  #  * For licensing information check the above url.
10 #  * Please do not remove this header.
11 #  * */
12
13 import pickle
14
15 """
16 For the data file manipulations in this module,
17 see the answer in:
18   https://stackoverflow.com/questions/4529815/saving-an-object-data-persistence
19 """
20
21 class TemporaryContainer(object):
22   """
23   TemporaryContainer
24   (Mon Jul 29 17:04:52 AST 2019)
25
26     Stored data will appear as member variables of the
27     instances of this class.
28   """
29   pass
30
31 class tmpcont(TemporaryContainer):
32   """
33   tmpcont(TemporaryContainer)
34   (Mon Jul 29 17:04:52 AST 2019)
35
36     Alias of TemporaryContainer for convenience.
37   """
38   pass
39
```

"pickle_file_manip.py" modülü içerisinde dosya kayıt veya dosya içerisinden nesne çekme işlemlerini yapmamızı sağlayacak fonksiyonlar ve yine kaydedilecek veya çekilecek objeyi oluşturacak bir class yapısına ihtiyaç vardır.

1.satır bir "shebang" satırıdır komut satırına "which python" yazıp bu shebang satırını elde edebiliriz. Python'a hangi PATH ile erişebileceğini script'e bildiriyoruz, her şeyi usr/bin altındaki python interpreter'ini kullanarak yap diyoruz aslında ve 1.satırda olması şart.

21.satırda bir class tanımı vardır (Base Class) TemporaryContainer(object) class'ından daha sonra türeyecek olan class olacaktır. (object) ifadesi bu class'dan türeyecek olan class'lara bazı özellikler kazandıracaktır. Class'ın içinde hiçbir şey yapılmadığından pass denilip derleyicinin hata vermemesi sağlanacaktır.

31.satırda yukarıdaki class'dan türeyen bir tmpcont(TemporaryContainer) class'ı görüyoruz dosyaya kaydedebilmek için oluşturulacak nesneler bu class üzerinden oluşturulacaktır içerisinde hiçbir şey olmamasına rağmen TemporaryContainer'ın kazandırdığı bazı özellikler vardır ve ilerleyen aşamalarda obj = tmpcont() olarak bir obje oluşturarak bu objeye member variable kazandırabileceğiz direkt olarak obj.number = 5 vs. gibi şeyler yazıp bu class'ın bu instance'sine member variable kazandırabiliriz (kazandıracağız).

```
40 def store(path_file, stored):
41   """
42     store(path_file, stored)
43     (Mon Jul 29 17:04:52 AST 2019)
44
45       Stored data in 'stored' in a file whose path is given
46       as 'path_file'.
47   """
48   with open(path_file, 'wb') as output:
49     pickle.dump(stored, output, pickle.HIGHEST_PROTOCOL)
50
51 def pickled_items(path_file):
52   """ Unpickle a file of pickled data. """
53   with open(path_file, "rb") as f:
54     while True:
55       try:
56         yield pickle.load(f)
57       except EOFError:
58         break
```

40.satırda başlayan store fonksiyonu class'ın member fonksiyonu değildir bir global fonksiyondur. Input parametresi olarak ilk olarak dosyanın kendi PATH'ini (örneğin documents/load.pkl şeklinde) almaktadır, ikinci parametre ise muhtemelen bir class instance olacaktır yani dosyaya kaydedilmek istenen şeyi almaktadır.

48.satırda "with" keyword'ü birçok şeyi tek başına yapmaktadır. İlgili dosyanın PATH'ine gidip dosyayı açıyor (dosya yok ise oluşturur) yazma işlemi bittikten sonra yani fonksiyondan çıktıktan sonra otomatik olarak dosyayı kapayacaktır. 'wb' demek write dosyaya bir şeyler yazacağımızı söylüyoruz. Aynı zamanda dosyayı açma yetkisi yok ise exception atıp program kesilecektir veya yazılmak üzere bir dosya oluşturulmamışsa with yine bu dosyayı açamayacaktır.

49.satıda pickle.dump() fonksiyonu ile kaydedilecek olan nesne dosyaya dökülüyor, bunu yaparken nesneyi başka bir script'de çağırabileceğimizden nesne ile alakalı tüm bilgileri de kaydediyor. Fonksiyon sırası ile kaydedilecek nesne ardından hangi dosyaya kaydedileceği (bu alias kullanarak output olarak belirlenmiş), son olarak dosya kayıt protokolü almaktadır.

51.satırda başlayan pickled_items fonksiyonu input parametresi olarak okunacak dosyanın PATH'ini almaktadır.

53.satır bir önceki fonksiyondan farklı olarak okuma işlemi yapılacağından "rb" olacaktır.

54.satıda bir sonsuz döngü başlatıyoruz ve gidip dosya içerisinden nesneleri teker teker çekmeye çalışıyoruz. try catch yapısı ile beraber kullanılan yield keyword'ünün amacı;

- Yield aracılığı ile pickle içeindeki datalar teker teker alınıyor içinde ne bulmuş ise binary formattan çekip iterasyon içinde gönderecek.

- Tüm dosyayı dolaştı her şeyi gönderdi dosya sonuna geldi ve bir exception ile karşılaştı EOFError ile bir exception atılmış fakat bu yakalanmamış yakalanmayan execption programın sonlanmasına sebep olur bunu önlemek için break ile while true'dan çıkıyoruz while'dan çıkmak beraberinde fonksiyonu da sonlandıracaktır.

- pickle.load(f)'in görebi f dosyası (ilgili path'e sahip ve görevi read olan bir alias) içerisindeki nesneler bu formatta load fonksiyonu aracılığı ile tek tek alınabiliyor. Dolayısı ile bir yield gerekli oluyor.

13.1.1.2 pfm_test.py

In Python:

```
#!/usr/bin/python

# /*
#  * This file is part of the "dev-in-place" repository located at:
#  * https://github.com/osuvak/dev-in-place
#  *
#  * Copyright (C) 2020  Onder Suvak
#  *
#  * For licensing information check the above url.
#  * Please do not remove this header.
#  * */

import getopt, sys
import os
import pickle_file_manip as pfm
import numpy as np

class DataComputerHolder(object):
  """
  DataComputerHolder(object)
    Class for holding computed data (illustrative).
  """
  def __init__(self, number=10):
    self.content \
      = np.cumsum( np.ones(number) )
```

DataComputerHolder class'ının görevi 13.1.1.1'deki tmp_cont class'ına member kazandırmaktır. Bu class içerisi doldurulmuş, boost_bimap örneğinde olduğu gibi burada da default constructor ve constructor aynı yapı içerisinde oluşturulmuştur. __init__ C++'daki consturctor'un yaptığı işi yapmaktadır. Instance'ye bir ilk değer kazandırıyor. İlk değer olarak default olarak yapılan şey şudur;

Aslında bir array' içerisindeki elemanları toplaya toplaya gidiyor; array olarak 10 uzunluğunda birlerden oluşan bir array tanımlanmış.

Sonuç olarak 1, 2, 3, 4, ... 10 olacaktır son eleman 10 çünkü son index'e gelene kadar toplam 9 tane 1 var 9*1 + 1 diyerek düşünürsek 10 olmalı. Buradaki array'in sahip olacağı 1 sayısı değiştirilebilir fakat kodda değiştirilmemiş sonuçta basit bir örnek gösterilmek isteniyor.

```
action = None
try:
  opts, args = \
    getopt.getopt( sys.argv[1:],
                   "a:", ["action="] )
except getopt.GetoptError as err:
  print str(err)
  sys.exit(-1)
```

action = None değişkeni noneType bir değişkendir bu action'a belirli değerler verilmesi üzerinden dosyadan okuma işlemi mi yoksa, dosyaya yazma işlemi mi yapılacaktır? Sorusunun cevabını bu değişken tutacaktır.

getopt.getopt() fonksiyonu kurala dayalı komut satırı seçenekleri için bir ayrıştırıcıdır. Genelde sys.argv gibi bir bağımsız değişken dizisini ayrıştırmak için kullanılır diğer bir deyişle, bu modül (getopt) komut dosyalarının sys.argv'deki komut satırı argümanlarını çözümlemesine yardımcı olur.

try – catch yapısı şu sebeple kullanılıyor kodu derlerken komut satırından girilecek bir bilgiyi tutuyor örneğin aşağıdaki komut satırına bakacak olursak;

```
soray@soray-VirtualBox:~/Desktop/433_hw/hw_2$ python3 pfm_test.py -a load
what is inside sys.argv[1:] ->  ['-a', 'load']
what is inside opts :       ->  [('-a', 'load')]
```

Ilgili yerlere print'ler koyulmuştur hangi veri hangi tipte tutulacağını gözlemlemek için, bu şekilde bir compilation yaparsak sys.argv[1:] script isminden SONRA gelen bilgiyi tutar. Script ismini sys.argv[0] tutuyor.

Sys.argv[1:] == [‘-a’, ‘load’] bilgisini tutuyor bu bilgi komut satırından verildi. Bu bilgi opts değişkeni içerisine kaydediliyor kaydedilirken tuple tipinde olduğu gözüküyor. Bu veri bir sonraki kısımda kullanılacak ve noneType olan action bir anlam kazanacak. Exeption atılıp eğer komut satırına verilecek bilgi girilmez ise programın sonlanması isteniyor. (Exeption yakalanmıyor)

31.satır’da “a:”, [“action=”] burada “:” kullanılmasının sebebi komut satırından girilecek -a load olacağı için a’nın arkasından bir şeyler yazılacağını bildiriyoruz. Action ise a’nın uzun ismi olarak tanımlanıyor.

```
35
36 for o, a in opts:
37   if o in ("-a", "--action"):
38     action = a
39   else:
40     pass
41
```

35.satırda opts tuple’ının içerisine gidiliyor. Tuple ne tutuyordu? (‘-a’, load) dolayısıyla o = -a ve a = load gibi düşünebiliriz.

37.condition eğer o bir ‘-a’ veya ‘—action’ ise yani istenilen formatta bir komut satırı bilgi aktarımı sağlanmış ise a değeri yani load’ı action’a ata işlemi yapılmıştır. Aşağıda action kullanarak da bir komut satırı girişi yapabileceğimizi görüyoruz. Else durumda bir şey yapma zaten daha önceki işlemler ile komut satırına girilmesi gereken keyword’lerin a veya action olması garantilendi. (28.satıdaki yapıda garantilendi)

NOT : Buraya kadar makefile’dan bahsedilmedi daha sonra bahsedilecek.

```
soray@soray-VirtualBox:~/Desktop/433_hw/hw_2$ python3 pfm_test.py -a load
what is inside sys.argv[1:] ->  ['-a', 'load']
what is inside opts :       ->  [('-a', 'load')]
what is inside action :     ->  load
Content of Object Loaded :
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
soray@soray-VirtualBox:~/Desktop/433_hw/hw_2$ python3 pfm_test.py --action load
what is inside sys.argv[1:] ->  ['--action', 'load']
what is inside opts :       ->  [('--action', 'load')]
what is inside action :     ->  load
Content of Object Loaded :
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

```
41
42 dname_data = 'data_folder'
43 fname_data = 'pickle_file.pkl'
44 path_file_data = '%s/%s' % (dname_data, fname_data)
45 os.system( 'mkdir -p %s' % dname_data )
46
```

42. ve 43.satırlarda oluşturulacak dosyanın path’i belirleniyor oluşturuluyor.

44.satırda iki PATH birbiri ardına eklenerek / işareti ile ilgili directory’nin altında olduğunu söylememiz sağlanıyor. Sonuç olarak path_file_data = data_folder/pickle_file.pkl olarak anlam kazanıyor bu bir PATH’dir. Hangi Path? Nesnelerin kaydedileceği veya kaydedilen nesnelerin dosyadan çekilebileceği dosyanın PATH’i.

45.satırda os.system kullanılarak komut satırında yapılabilen işlem python script’inin içinde yapılıyor (mkdir -p) gidip script’in olduğu directory’de bir directory oluşturuyor (zaten varsa bir şey yapmıyor yoksa oluşturuyor) oluşturduğu directory’e isim olarak dname_data ismini veriyor.

```
47 if   action == 'store':
48   oa = DataComputerHolder()
49   print "Content of Object to be Stored :"
50   print oa.content
51
52   out = pfm.tmpcont()
53   out.oa = oa
54   stored = [ out ]
55   del out
56   pfm.store(path_file_data, stored)
57
```

47.satırdan başlayan if yapısı action'a bakarak bir karar veriyor (action'un nasıl belirlendiğini yukarıda çözümlemiştik bize bağlı bizim ne yapmak istediğimize bağlıdır), eğer action == 'store' ise dosyaya nesnelerin kaydedileceği store case'i gerçekleşecektir.

48.satırda bu script'in en başında tanımlanan class yapısının bir instance'si oluşturuluyor default olarak oluşturulmuş (yani 10 tane 1 olan array). 49 ve 50. Satır çalıştığında kümülatif toplam yapılan bir yapının yansıtılmasını bekliyoruz, yani 1, 2, 3, 4, 5 . . . . 10 ekrana bastırılacak bu değişkenin tipi ndarray'dir.

52.satırda 13.1.1.1'de tmpcont class'ının objelerini dosya içerisine kaydedeceğimizi söylemiştik ama bu class içerisinde hiçbir şey yapılmıyordu pass denilip çıkılmıştı.

52.satırda out isminde pfm.tmpcont() class'ının instancesi oluşturulmuştur.

"pfm nedir?"

15.satırda import ettiğimiz modül olan 13.1.1.1 modülünün bir alias'ıdır. Bu alias kullanılarak class oluşumu sağlandı yani demek istenen şu pfm alias'ına sahip modüle gir ve tmpcont() class'ını bul ve oluştur.

53.satırda out objesine member variable kazandırılmıştır aslında tam olarak tmpcont class'ına DataHolder class'ının instance'si kazandırılmıştır, "oa" içerisinde bir content var bu content cumsum'un yaptığı şeyi tutuyor content aslında bir member variable for DataComputerHolder diyebiliriz.

Dosya içerisine content kaydedilmiyor ilgili content'e sahip DataHolder class'ının objesinin adresi tmpcont class'ına kazandırılıyor (member variable olarak) ve bu adres dosya içerisine store'ediliyor. Bu adres 54.satırda listeye çevrilmiş ve out'un kendisi yok edilmiştir. İlgili adrese sahip oluyoruz artık o adresin gösterdiği out'a ihtiyacımız yok onu siliyoruz.

Son aşamada stored içerisinde DataHolder classının bir instancesi var ve bu instance tmpcont'a member variable olarak kazandırılmış. Dolayısı ile tmpcont'un objesi dosyaya kayıt edilecektir bu objenin geldiği yer DataHolder class'ıdır.

pfm alias'ına sahip olan 13.1.1.1 modülü içerisinde store fonksiyonu kullanılıyor daha önce bu store fonksiyonun yaptığı işi açıklamıştık kısaca tekrar edersek verilen dosya PATH'indeki dosyaya class objesini binary formatta dump ediyor kaydediyor.

```
soray@soray-VirtualBox:~/Desktop/433_hw/hw_2$ python3 pfm_test.py --action store
what is inside sys.argv[1:] ->  ['--action', 'store']
what is inside opts :       ->  [('--action', 'store')]
what is inside action :     ->  store
Content of Object to be Stored :
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
oa object  : -> <__main__.DataComputerHolder object at 0x7f8733e28f10>
out object : -> <pickle_file_manip.tmpcont object at 0x7f8733eb9220>
```

Burada out objesinin ne olduğu gözükmektedir, tmpcont class'ı içerisine bir (<<) operatörü yani (__str__) operatörü overload edilse idi print(out) yazdığımızda print(oa.content) ile aynı çıktıyı elde edecektik.

```
57
58 elif action == 'load':
59   if not os.path.isfile(path_file_data):
60     raise NameError('Cannot find %s.' % path_file_data)
61
62   for item in pfm.pickled_items(path_file_data):
63     for tmp in item:
64       print "Content of Object Loaded :"
65       print tmp.oa.content
```

58.satıda reading case'i gerçekleşecektir. Dosya içerisinden objeleri çekme işlemi yapılacaktır.

59. ve 60.satırlarda bir check yapılıyor ilgili PATH'de yani dosyanın olması gereken path'de dosya var mı? Kontrolü yapılıyor yok ise error verilip programın sonlanması sağlanıyor.

62.satırda iteratif olarak dosya içerisinde nesneleri çekmek istiyoruz bunu yaparken yield özelliğini kullancağız. For yapısında in pfm.pickled_items(PATH) yapısı kullanılarak iteratif olarak item'a ne kadar nesne var ise sırası ile atanacaktır.

İtem'ın tipi nedir ? Cevap Liste çünkü dosya içerisine kaydedilen objeler liste tipinde bir member variable'lara sahipti 54.satırda bu böyle belirlendi.

Item dosya içerisinden çekilen ilk listeyi tmp'ye tutturur ikinci for sadece 1 iterasyon yapacaktır çünkü liste içerisinde sadece adres var hangi adres oa'nın adresi print(tmp.oa.content) bize o uzun kümülatif toplamı verecektir.

Output:

```
soray@soray-VirtualBox:~/Desktop/433_hw/hw_2$ make clean
soray@soray-VirtualBox:~/Desktop/433_hw/hw_2$ make action_store
Content of Object to be Stored :
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
soray@soray-VirtualBox:~/Desktop/433_hw/hw_2$ make action_load
Content of Object Loaded :
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

13.1.1.3 makefile for pfm_test.py

```
1  # /*
2  #  * This file is part of the "dev-in-place" repository located at:
3  #  * https://github.com/osuvak/dev-in-place
4  #  *
5  #  * Copyright (C) 2020  Onder Suvak
6  #  *
7  #  * For licensing information check the above url.
8  #  * Please do not remove this header.
9  #  * */
10
11 # Updates
12 #   Commented through with interspersed questions on
13 #
14
15 script_name := pfm_test.py
16 path_data   := data_folder
17
18 default: help  force_look
19
20 action_store : force_look
21         @./$(script_name) -a store
22
23 action_load : force_look
24         @./$(script_name) -a load
25
26 clean : aux_clean_pyc  aux_clean_dir  force_look
27
28 aux_clean_pyc : force_look
29         @find . -name "*.pyc" -type f -delete
30
31 aux_clean_dir : force_look
32         @mkdir -p $(path_data) && cd $(path_data) && rm -rf ./*
33
34 help : force_look
35         @echo ""
36         @echo "*** RECIPES ***"
37         @echo "  default - helps"
38         @echo "  store   - stores data"
39         @echo "  load    - loads data"
40         @echo "  clean"
41         @echo "  help"
42
43 force_look :
44         @true;
```

Makefile aracılığı ile kodu çalıştırmak daha kolaydır beraberinde getirdiği recipy'ler hayatımızı kolaylaştırmaktadır.

15. ve 16.satırda dosyanın tutulduğu directory ismi be script ismi tanımlanmıştır. := syntax'ı yerine başka alternatifler de vardır.

default recipy'si makefile içinde yazılan help recipy'sini çağırmaktadır.

action_store recipy'si script_name ismine sahip olan script'i -a store komut satırı vererek çalıştırmaktadır. -a store yerine –action store yazılsaydı da çalışacaktı. Yani dosyaya yazma işlemini gerçekleştiren if condition'u işini yapacaktır.

action_load recipy'si ise store recipy'sine benzer olarak if condition'u dosyadan nesne çekmek olan yapıyı işletecektir aslında tüm kod çalışacak ama bu if'e girilecektir.

Clean recipy'si 2 farklı recipy'i arka arkaya gerçekleştirecektir. Bunlar aux_clean_pyc kod çalıştırıldıktan sonra binary .pyc uzantılı dosyalar belirmektedir bu dosyaları silmek için bu recipy yazılmıştır @find ile ".pyc" uzantılı dosyaları veya directory'leri bul işlemi yapılıyor -delete ile de bu bulunanlar siliniyor. aux_clean_dir recipy'si ise dosyanın tutulduğu directory olarak bildiğimiz data_folder directory'sini sırası ile yoksa oluştur , bulduktan sonra içine gir, girdikten sonra bu directory'i sil işlemleri yapılmaktadır. Bunların hepsi tek satırda olması şart ayrı ayrı satırlarda olursa makefile kendisini ve hatta script'i silebilir.

13.1.2 chunk-management-example-octave

13.1.2.1 chunks.m

In Octave:

```
1  %{
2  /*
3   * This file is part of the "dev-in-place" repository located at:
4   * https://github.com/osuvak/dev-in-place
5   *
6   * Copyright (C) 2020  Onder Suvak
7   *
8   * For licensing information check the above url.
9   * Please do not remove this header.
10  * */
11 %}
12
13 close all
14 clear all
15
16 no_items_per_chunk =  100;
17 no_total            = 5346;
18
19 cnt_chunk         = 0;
20 cnt_current_chunk = 0;
21
22 for kk=1:no_total
23     cnt_current_chunk += 1;
24     lval = (cnt_current_chunk >= no_items_per_chunk) ...
25         || (kk >= no_total);
26     if lval
27         cnt_chunk        += 1;
28         disp( sprintf('New Chunk (%6d) has %6d items' , ...
29             cnt_chunk, cnt_current_chunk) );
30         cnt_current_chunk = 0;
31     end
32 end
```

13.1.2.1 kapsamında bir algoritma kurularak gösterim şekli gösterilmiştir no_total kadar olan veriler 100'erli olarak gruplanmış ve son grup 46 olmak zorunda kalmıştır. Bu gruplamayı ve son grubun sorun çıkarmadan 46 tane item alabilmesi sağlanmıştır.

13. ve 14.satırda her octave kodunda yazılması şart olan satırlardır önce close sonra clear sıralaması olması şarttır. Kodda kullanılan değişkenlerin yapıların silinmesi şarttır silinmeden yapılırsa bir önceki compile sonucu elde edilen değerler bir sonraki compile sonucunda sorun yaratabilir.

16. ve 17.satırda yukarıda bahsedilen kapsamda her bir chunk için kaç tane item alabileceğini söylüyoruz ve toplam item sayımız no_total'a atanmıştır.

19. ve 20.satırlardaki değişkenler for içerisinde cnt_current_chunk kullanımı chunk'un 100'lük alanı dolduğunda if'e girmemizi sağlayan diğeri ise toplam kaç tane chunk oluşturulduğu bilgisini tutmaktadır. Her bir 100'lük chunk dolduğunda cnt_chunk +1 artacaktır. Son case bir istisna neden istisna aşağıda açıklanacaktır.

For 1'den no_total'e kadar gidebiliyor, cnt_current_chunk başlıyor saymaya 1, 2, 3 .. . .. 100 olduğunda bir chunk için gerekli item sayısı olan sayıyı buluyoruz.

Lval veya ile iki condition'a bağlıdır, o veya bu gerçekleşirse True olacaktır dolayısı ile if'e girebileceğiz. İlk condition cnt_current_chunk bir chunk için belirlenen item sayısı >= 100 oldu mu? Cevap evet ise Lval = TRUE oluyor 2.condition'a bakmaya gerek yok zaten TRUE oldu.

Cnt_current_Chunk 100 oldu if yapısı çalışıyor. If içerisinde yapılan şey ilk olarak chunk sayısını 1 arttırmak oluyor yani 100'lük itemi tutacak ilk chunk'a sahibiz. Ardından disp ve sprintf yapısı kullanarak değişkenleri bastırıyoruz.

Sırası ile kaçıncı chunk olduğu ilk iterasyonda bu 1'dir, ardından bu chunk içinde kaç tane item var istenildiği gibi 100 bastırılacaktır bu işlem yapılırken c++'daki setw(6)'ya benzer bir yapı kullanılmış. If'den çıkmadan cnt_current_chunk = 0 yapılıyor ve bir sonraki iterasyonlarda yine aynı iş yapılacaktır cn_current_chunk saymaya başlayacak 100 olunca if'e girilecek bu sefer cnt_chunk bir daha artıp 2 olacaktır.

Bu işlem 53 kere 100'er itemden oluşan chunk oluşturacak 54.olarak yani son iterasyonda;

Lval'de ilk condition false ikinci condition TRUE olacak yani son item'a ulaşmış oluyoruz elimizde ne kaldıysa onu tutup o kadar bir item tutacak chunk'a atmak zorundayız. Dolayısıyla son chunk 46 item tutacaktır. Lval'e 2 ihtimal vererek son iterasyonda hata almamayı sağlıyoruz.

Output:

```
Command Window
New Chunk (    13) has    100 items
New Chunk (    14) has    100 items
New Chunk (    15) has    100 items
New Chunk (    16) has    100 items
New Chunk (    17) has    100 items
New Chunk (    18) has    100 items
New Chunk (    19) has    100 items
New Chunk (    20) has    100 items
New Chunk (    21) has    100 items
New Chunk (    22) has    100 items
New Chunk (    23) has    100 items
New Chunk (    24) has    100 items
New Chunk (    25) has    100 items
New Chunk (    26) has    100 items
New Chunk (    27) has    100 items
New Chunk (    28) has    100 items
New Chunk (    29) has    100 items
New Chunk (    30) has    100 items
New Chunk (    31) has    100 items
New Chunk (    32) has    100 items
New Chunk (    33) has    100 items
New Chunk (    34) has    100 items
New Chunk (    35) has    100 items
New Chunk (    36) has    100 items
New Chunk (    37) has    100 items
New Chunk (    38) has    100 items
New Chunk (    39) has    100 items
New Chunk (    40) has    100 items
New Chunk (    41) has    100 items
New Chunk (    42) has    100 items
New Chunk (    43) has    100 items
New Chunk (    44) has    100 items
New Chunk (    45) has    100 items
New Chunk (    46) has    100 items
New Chunk (    47) has    100 items
New Chunk (    48) has    100 items
New Chunk (    49) has    100 items
New Chunk (    50) has    100 items
New Chunk (    51) has    100 items
New Chunk (    52) has    100 items
New Chunk (    53) has    100 items
New Chunk (    54) has     46 items
>>
```

13.1.2.2 inputs_for_chunks.m

In Octave:

```
%{
/*
 * This file is part of the "dev-in-place" repository located at:
 * https://github.com/osuvak/dev-in-place
 *
 * Copyright (C) 2020  Onder Suvak
 *
 * For licensing information check the above url.
 * Please do not remove this header.
 * */
%}

function str = inputs_for_chunks()
    str.path_data          = 'data_folder';
    str.name_mat_common    = 'data';
    str.no_items_per_chunk = 100;
    str.no_total           = 5346;

    str.no_snippet_data    = 3;
%      str. =
end
```

Burada bazı değerler belirlenmiş ve bir void fonksiyon'a kaydedilmiştir. (Struct yapısı da olabilir) sonuç olarak storage veya load yapan script'lerde bu fonksiyonun kullanım amacı içerisindeki bilgileri 2 ayrı script'e de sağlamak. Buradaki değişkenler input her iki script için de input parametreleridir.

Bu değişkenler sırası ile;

(str.str_path_data = 'data_folder') : ".mat" uzantılı dosyaların hangi directory altına kaydedileceğini veya bu dosyalar koda çağırılarak okunacağı için hangi directory içerisindeki dosyalar çekilecektir sorusunun cevabını söylemektedir.

(str.str_name_mat_common = 'data') : Her bir dosyanın hangi isim ile isimlendirileceği bilgisini tutmaktadır.

(str.no_items_per_chunk = 100) : Her bir chunk içerisinde kaç tane item (değer) olacaktır ? 13.1.2.1'deki ile aynıdır her 100 item'da bir yeni bir dosya oluşturulacaktır!

(str.no_total = 5346) : Toplam kaç tane kaydedilecek item var? Bu itemlar 100'erli 100'erli ".mat" uzantılı dosyalara kaydedilecekler.

(str.no_snippet_data = 3) : Bu değişken Load script'i için özel bir değişkendir 3'tane baştan 3'tane sonran item al ve onları bastır derken kullanılacaktır.

13.1.2.3 chunks_with_storage.m

In Octave:

```
1  %{
2  /*
3   * This file is part of the "dev-in-place" repository located at:
4   * https://github.com/osuvak/dev-in-place
5   *
6   * Copyright (C) 2020  Onder Suvak
7   *
8   * For licensing information check the above url.
9   * Please do not remove this header.
10  * */
11 %}
12
13 close all
14 clear all
15
16 %  inputs
17 str = inputs_for_chunks();
18
19 path_data          = str.path_data;
20 name_mat_common    = str.name_mat_common;
21
22 no_items_per_chunk = str.no_items_per_chunk;
23 no_total           = str.no_total;
24
25 %  script
26 cnt_chunk         = 0;
27 cnt_current_chunk = 0;
28
29 cnt_store = 0;
30 to_be_stored = [];
31
```

17.satıda 13.1.2.2'de yazılan struct yapısı buraya input olarak verildi, str içerisinde ihtiyacımız olan değerler yeni değişkenlere atandı bu atama olmasa da olurdu fakat her yere str. Yazmak gerekecekti, bunu istemeyeceğimizden path_data = str.path_data gibi yeniden isimlendirme kullanılmıştır, bu da beraberinde ekstra kopyalamalar getirmiştir (Matlab'da call by reference olmadığından kopyalamak zorundayız)

26 ve 27.satırlar 13.1.2.1'de olduğu gibi for içerisinden if içerisine geçebilmemizi sağlayan değişken cnt_current_chunk'dır, cnt_chunk ise toplam kaç tane chunk(dosya) oluşturuldu bu bilgiyi tutmaktadır.

cnt_store kullanım amacı?

Burada aşağıdaki yapı incelendiğinde cnt_store 1'den no_total'a kadar değer alacaktır bu değerler aslında item'lardır. Item'lar tam olarak nedir? ".mat" uzantılı dosyalara kaydedilecek olan değerlerdir bu değerler 100 uzunluklu array'lere tututrulacak ve ardından bu array dosyaya kaydedilecektir. Örneğin ilk dosyaya 1'den 100'e kadar olan itemler kaydedilecektir, 2.dosyaya 100'den 200'e kadar olan itemler kaydedilecektir bu şekilde ilerleye ilerleye gerçekleşecek bir kayıt söz konusudur.

1'den 100'e kadar olan itemler nerede tutulacak?

To_be_stored = [] matrix'i (yukarıda array dendi ama aslında bir double matrix) tam olarak 1.iterasyon yani 1.dosya için bu işi yapmaktadır aslında tüm dosyalar için 100 tane item tutacaktır, her bir dosya kaydı sonrasında bu değişkenin tekrar sıfırlanması şarttır.

Burada bahsedilen her item aslında bir double tipinde değişkendir ve matrix içinde kendisine yer bulacaktır.

```
31
32 for kk=1:no_total
33     cnt_store += 1;
34 %      to_be_stored = [ to_be_stored  cnt_store ];
35     to_be_stored(end+1) = cnt_store;
36
37     cnt_current_chunk += 1;
38     lval = (cnt_current_chunk >= no_items_per_chunk) ...
39         || (kk >= no_total);
40     if lval
41         cnt_chunk        += 1;
42         disp( sprintf('New Chunk (%6d) has %6d items' , ...
43             cnt_chunk, cnt_current_chunk) );
44         cnt_current_chunk = 0;
45
46         system( sprintf( 'mkdir -p %s' , path_data ) );
47         save( sprintf( ...
48             '%s/%s_%016d.mat', ...
49             path_data, name_mat_common , cnt_chunk) , ...
50             'to_be_stored' );
51         to_be_stored = [];
52     end
53 end
```

For yapısına giriyoruz toplam no_data kadar item var ve bu item'ların değeri ise 1'den no_data'ya kadardır 100'er 100'er dosyalara kaydedileceklerdir son iterasyon yine farklı olacaktır.

33.satır ile 36.satır arasına bakalım cnt_current_chunk henüz yüz olamamış durumu değerlendirelim, benim item olarak adlandırdığım şeyler aslında birer cnt_store'dur, cnt_store 1'den 5346'ya kadar değerler olabilecektir.

35.satırda to_be_stored(end+1) = cnt_store ilk 100 item için to_be_stored = 1'den 100'e kadar olan sayılar olacaktır, to_be_stored aslında 100'lük item dizisini (bu yanlış tanım 1x100 lük bir matrixtir) tutmaktadır ve bu dizi zaten dosyaya kaydedilecek olan veridir.

100-200 arasındaki itemları yine to_be_stored(end+1) ile yapabiliyoruz çünkü ilk 100'lük veriyi kaydettikten sonra to_be_stored = [] yapılarak boşaltılıyor ve end'de aynı şekilde sıfırlanıyor if'den çıktık bir end ile karşılaştık end artık sıfırlandı.

37.ve 38.satırlar daha önce 13.1.2.1'de bahsedildiği gibidir iki ihtimalli condition (veya) yapılmaktadır. Bunu yapmamız şart çünkü son item dizisi 100 tane item'dan oluşmayabilir!

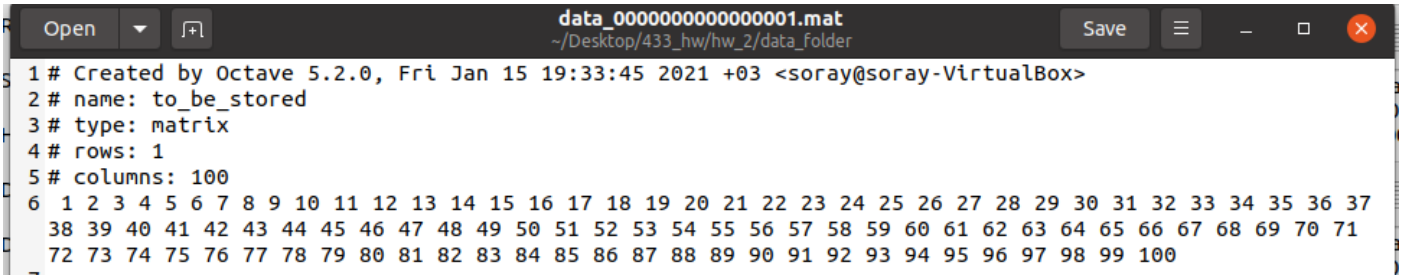
Lval TRUE olduğunda ekrana istenen değerler basılacaktır cnt_current_chunk'ı sıfırlamamız şart 100'lük itemlar bunu sıfırlamazsak oluşmaz.

46.satırda python'daki os.system'e benzer bir yapı var görevi komut satırına bir şeyler yazmak ("mkdir -p %s", path_data) görevi path_data path'ine sahip bir directory var mı? Yok ise oluşturmaktır, var ise birşey yapmamaktır.

47.satırda save fonksiyonu kullanılarak to_be_stored içinde ne var ise istenilen path_data'nın tuttuğu PATH'e bir dosya açıp bu dosyaya kaydetmektir. Aynı zamanda oluşturulacak dosyanın ismi name_mat_common yani "data" ile başlayacaktır ek olarak bu isim'e cnt_chunk bilgisi de eklenecektir fakat %016d dendiğinden isim şu şekilde olacaktır; data000….0001 buradaki en sağdaki bir cnt_chunk'ı ifade ediyor ve onun solunda toplam 15 tane sıfır vardır, dosya isimlerinin düzenli ve sıralı olması istendiğinden bu yapı tercih edilmelidir.

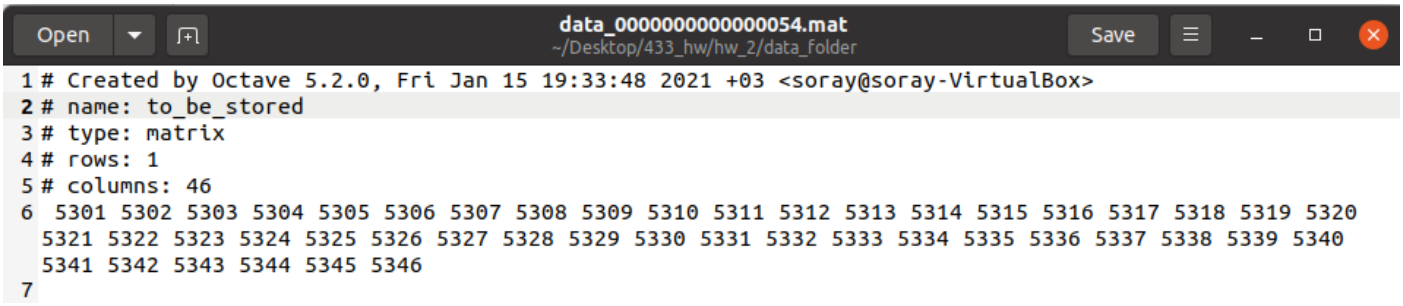
Save fonksiyonu 2 parametre alır sol taraftaki parametre şudur; sprintf(%s%s%016d.mat, ……) bu parametre sprintf bir string return edeceğinden aslında ilk parametre bir dosya ismi parametresi

olmaktadır. Sprintf kullanarak güzel kullanışlı bir isim belirlendi, İkinci parametre ise dosyaya ne kaydedilecek sorusunun cevabını tutmaktadır ‘ ’ arasına yazılmak durumundadır. Dosyaya kaydedilecek olan şeyin 100 tane item’dan oluşan ve bu itemlar 1’den 100’e kadar olan itemlar olduğunu 1.dosya için biliyorduk dolayısı ile dosyayı açıp incelersek aşağıdakini görebiliyoruz.

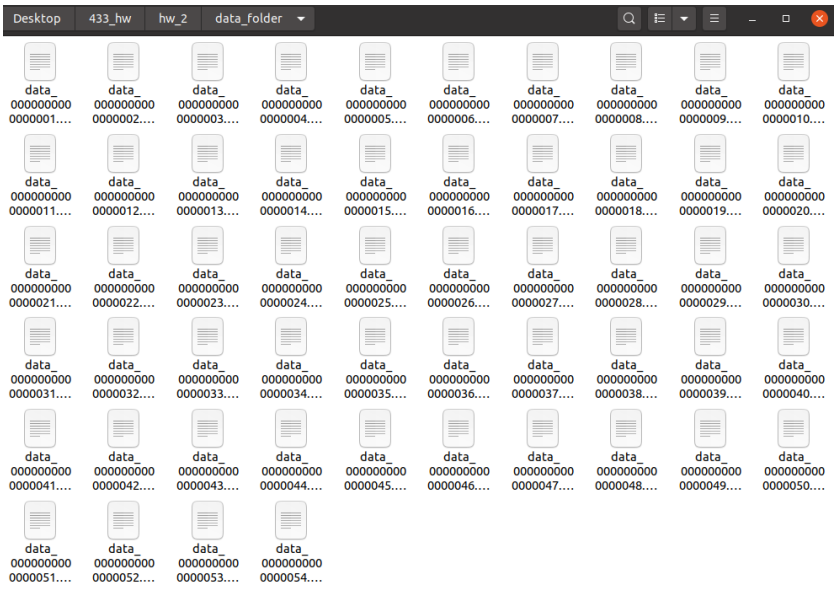


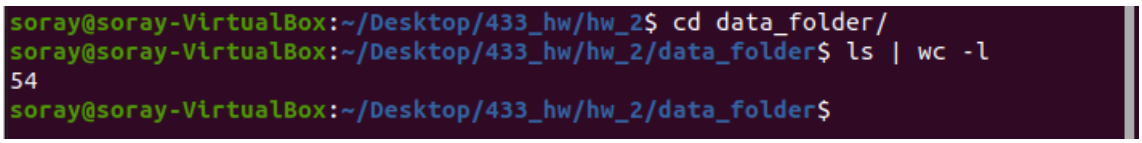
İkinci dosya 100+1’den 100+1+99’a kadar olan itemleri tutacaktır. Bu şekilde tüm 100 uzunluklu diziler dosyalara kaydedilecektir.

Son iterasyonda Lval yapısı içinde 2.condition işleyecektir neden? 100 tane item kalmadı bundan az sayıda item var ne kadar var ise o kadarını kaydetmemiz lazım. Son iterasyon sonucu da şu şekilde bir kayıt sağlamıştır:



Colums : 46 Bu dosyada toplam 46 tane item olduğunu söylüyor bununla beraber save fonksiyonu bir çok veriyi de bu dosyanın içerisine kaydetmiştir.





Görüldüğü ve beklendiği gibi toplam 54 tane dosya oluşmuştur, 53 tanesi 1x100’lük matrix barındırıyor kalan 1 tanesi ise 1x46 boyutunda bir matrix barındırmaktadır.

13.1.2.4 chunks_with_loading.m

In Octave:

```
%{
/*
 * This file is part of the "dev-in-place" repository located at:
 * https://github.com/osuvak/dev-in-place
 *
 * Copyright (C) 2020  Onder Suvak
 *
 * For licensing information check the above url.
 * Please do not remove this header.
 * */
%}

close all
clear all

%  inputs
str = inputs_for_chunks();

path_data          = str.path_data;
name_mat_common    = str.name_mat_common;

no_items_per_chunk = str.no_items_per_chunk;
no_total           = str.no_total;

no_snippet_data    = str.no_snippet_data;

%  script
no_chunks = ceil(no_total / no_items_per_chunk);

```

23.satıra kadar olan kısım 13.1.2.3 ile aynıdır, 25.satırda gösterim yapılırken kullanılacak bir parametredir dosya içerisinde ilk 3 ve son 3 item gösterilecektir.

28.satırda no_chunks değişkeni toplam kaç tane chunks (dosya) vardır sorusunun cevabıdır buna bir bölme işlemi ile ulaşıyoruz 5346 / 100 = 53.4' tür bunu ceil ile yukarı yuvarlıyoruz yani no_chunks toplam chunks sayısı 54 olmuş oluyor bir önceki storage kodunda da 54 tane dosya oluşturulmuştu zaten.

```

for kk=1:no_chunks
    fname_mat = ...
        sprintf( ...
                 '%s/%s_%016d.mat', ...
                 path_data, name_mat_common , kk);

    % TODO: check if file with name fname_mat exists.

    load(fname_mat);
    disp( sprintf('Snippet from chunk %16d', kk) )
    if numel(to_be_stored) <= no_snippet_data
        disp(to_be_stored)
    else
        disp( [ to_be_stored(1:no_snippet_data) to_be_stored(end-no_snippet_data+1:end) ] )
    end
    disp('')
end
```

30.satırda artık toplam kaç tane chunks var ise o kadarlık bir iterasyon gerçekleşecektir. Kk = 1:54'e kadar gidebilecektir.

31.satırda fname_mat bir dosya adı tutacaktır 13.1.2.3'de oluşturulan dosyalara bir isimlendirme methodu kullanılmıştı aynı method burada da kullanılıyor ve ilgili path'deki dosyaların isimleri sırası ile dosyaları çağırılmak üzere fname_mat'a atanacaktır.

38.satırda load fonksiyonu fname_mat ismindeki dosyanın içindeki matrix'i koda çağırır bu matrix'in ismi to_be_stored'di. Ardından if else condition'ları sayesinde bu matrix'in ilk 3 ve son 3 item'ı gösterilmek üzere ekrana bastırılır.

Output:

```
..
   4101   4102   4103   4198   4199   4200

Snippet from chunk              43
   4201   4202   4203   4298   4299   4300

Snippet from chunk              44
   4301   4302   4303   4398   4399   4400

Snippet from chunk              45
   4401   4402   4403   4498   4499   4500

Snippet from chunk              46
   4501   4502   4503   4598   4599   4600

Snippet from chunk              47
   4601   4602   4603   4698   4699   4700

Snippet from chunk              48
   4701   4702   4703   4798   4799   4800

Snippet from chunk              49
   4801   4802   4803   4898   4899   4900

Snippet from chunk              50
   4901   4902   4903   4998   4999   5000

Snippet from chunk              51
   5001   5002   5003   5098   5099   5100

Snippet from chunk              52
   5101   5102   5103   5198   5199   5200

Snippet from chunk              53
   5201   5202   5203   5298   5299   5300

Snippet from chunk              54
   5301   5302   5303   5344   5345   5346
```

İstenildiği gibi her bir chunk'ın içindeki matrix'ten ilk 3 ve son 3 colum'undaki değişkenler bastırılmıştır.

13.1.2.5 makefile_for_invoking_scripts

```
# /*
#  * This file is part of the "dev-in-place" repository located at:
#  * https://github.com/osuvak/dev-in-place
#  *
#  * Copyright (C) 2020  Onder Suvak
#  *
#  * For licensing information check the above url.
#  * Please do not remove this header.
#  * */

path_data := data_folder

default : clean

clean : force_look
	@mkdir -p $(path_data) && cd $(path_data) && rm -rf ./*

force_look :
	@true
```

Burada basitçe clean recipy'sine sahip bir makefile yazılmıştır. Clean'in yaptığı iş data_folder var mı yok mu bak var ise bu directory'e gir ve bunun içindeki her şeyi silme işlemini yapmaktadır.