in computer sciences, a data structure is a logical structure meant to store data, in order to provide them an organization pthat would simplify their processing.

-- Wikipedia

ITI 1120
Module 6: Data Structure

General Concepts:

1. *Deepen the understandiing of data structures*

General Objective: *Solve problms that use data structures*

Learning Objective:

1. *Learn more about data structures (tuples, dictionnaires)*

Theme 1 *Deepen the knowledge of data structures*

Sub-theme: Composites types vs. simples

Composites data types (several values):

- Chain of caracters (ordered sequences of caracters, unchangeable), (ordered sequences of values of various types , changeable)
- Tuples (similai to a list but unchangeable)
- Dictionnairies (fast direct access, non-sequential, changeable)

Simple types (avalue): integer(int), real (float), boolean (bool) , byte sequences (bytes)

Sub-theme: Equality (==) vs. identity (is)

Lists:

```
>>> a = [1,2]
>>> a is [1,2]
False
>>> a == [1,2]
True

>>> b = [1,2]
>>> a == b
True
>>> a is b
False
```

Integers:

```
>>> x = 6
>>> y = 6
>>> x == y
True
>>> x is y
True
```

Chains of caracters

```
>>> x = “hello"
>>> y = “hello"
>>> x == y
True
>>> x is y
True
```

Sub-theme: Chaiîns of caracters (revisited)

Indexing: s[i] index I from 0 to len(s) -1

```
>>> s = "bonjour"
>>> s[0]
'b'
>>> s[6]
'r‘
```

Extraction of chain fragments: s[i:j] from index i to j-1

```
 >>> s[1:4]
'onj'
>>> s[:3]
'bon'
>>> s[2:]
'njour‘
```

The instruction **in** can be used independantly of **for**, to check if a given element *is among a sequence*

```
car = "e"
vowels = "aeiouyAEIOUYaaeeeeeuii"
if car in vowels:
  print(car, “is a vowel")
```

Visiting chains with while or for loops

```
>>> name ="Cleopatre"
>>> for car in name:
      print(car + ' *', end =' ')
C * l * e * o * p * a * t * r * e *

>>> name ="Josephine"
>>> index =0
>>> while index < len(name):
      print(name[index] + ' *', end =' ')
      # name[index] = 'x' # will cause an error
      index = index + 1
J * o * s * e * p * h * i * n * e *
```

Chains are unchangeable sequences

- We can not change a chain. If we add in the while loop

```
name[index] = 'x' # will cause an error
```

- We can change the variabke name to be a reference to another chain, we can not change the chain "Josephine"

```
name = name + 'xxx'

salut = ‘hello everyone'
salut[0] = 'B' # will cause an error
salut = 'B' + salut[1:] # a new chain
print(salut)

print(salut.upper()) # function from the str class
                     # produce a new chain
```

Sub-theme: byte sequences: type bytes

- Data of type bytes are very similar to data of type str, with a difference that they are strictly byte sequences, and not seuwences of Unicode caracters.
- They are used when we open files that do not contain text.

```
>>> s = "Amelie and Eugene\n"
>>> s
'Amelie and Eugene\n'
>>> octets = s.encode("Utf-8")
>>> octets
b'Am\xc3\xa9lie et Eug\xc3\xa8ne\n‘
>>> s.encode("Latin-1")
b'Am\xe9lie et Eug\xe8ne\n'
```

Chains are objects

- We can act on an objet using *methods* (functions associated with that object).

```
>>> c2 ="Vote for me"
>>> a = c2.split()
>>> print(a)
['Vote', ‘for', 'me']
>>> c4 = "This example, among others, can still serve"
>>> c4.split(",")
[‘This example', ‘among others’, ' can still serve']
>>> c2.find('me')
11
>>> c2.count('o')
2
>>>print(c2.index('o'))
1
```

Conversion of a byte chain into a str chain

```
>>> ch_car = octets.decode("utf8")
>>> ch_car
'Amelie and Eugene\n'
>>> type(ch_car)
<class 'str'>
>>> for c in ch_car:
      print(c, end =' ')

A m e l i e a n d E u g e n e
```

Sub-theme: Caracter chain format

- The tags to be usedare made of curly brackets that contain or not format indications.

```
>>> pi, r =3.1416, 4.7
>>> ch =« The area of a disk of radius {} is equal to {}."
>>> print(ch.format(r, pi * r**2))
The area of a disk of radius 4.7 is equal to 69.39794400000001
>>> ch =" The area of a disk of radius {} is equal to {:8.2f}.
    # 8 caracters, with 2 digits after the decimal point.
>>> print(ch.format(r, pi * r**2))
The area of a disk of radius 4.7 is equal to 69.40.

>>> coul =« green"
>>> temp = 1.347 + 15.9       # old fashion
>>> print ("Color %s and temperature %8.2f °C" % (col, temp))
Color green and temperature    17.25 °C
```

Sub-theme: Lists (revisited)

- To access a list elements, we use the same methodes (indexing and slicing)

```
>>> numbers = [5, 38, 10, 25]
>>> numbers[1:3]
[38, 10]
>>> numbers[:2]
[5, 38]
>>> numbers[1:]
[38, 10, 25]
```

- Check the occurrence of an element in a list with **in** :

```
n = 5
primes = [1, 2, 3, 5, 7, 11, 13, 17]
if n in primes:
    print(n, « is among the prime numbers")
```

Examples: List of caracter chains and list of various values

```
list = [dog', 'cat', 'crocodile', 'elephant']
for animal in list:
   print(‘size of the chain', animal, '=', len(animal))
```

- **L’execution donne :**

size of the chain dog= 3

size of the chain cat = 3

size of the chain crocodile = 9

size of the chain elephant = 8

```
divers = ['lézard', 3, 17.25, [5, 'Jean']]
for e in divers:
   print(e, type(e))
```

- **The execution gives:**

lezard <class 'str'>

3 <class 'int'>

17.25 <class 'float'>

[5, 'Jean'] <class 'list'>

The lists are changeable

```
>>> numbers = [5, 38, 10, 25]
>>> numbers[0] = 17
>>> numbers
[17, 38, 10, 25]
>>> numbers[1:3] = [40, 50] # changes 2 éléments
>>> numbers                 # from index 1 and 2
[17, 40, 50, 25]
```

Operations on lists, produce new lists

```
>>> fruits = ['orange',lemon']
>>> veggetables = [‘carotte','oignon','tomato']
>>> fruits + legumes
['orange', ‘lemon', ‘carotte', 'oignon', 'tomato']
>>> fruits * 3
['orange', ‘lemon', 'orange', ‘lemon', 'orange', ‘lemon']
```

Lists are objects

```
>>> numbers = [17, 38, 10, 25, 72]
>>> numbers.sort() # sort the list
>>> numbers
[10, 17, 25, 38, 72]
>>> numbers.append(12) # add an element at the end
>>> numbers
[10, 17, 25, 38, 72, 12]
>>> numbers.reverse() # reverse the order of elements
>>> numbers
[12, 72, 38, 25, 17, 10]
>>> numbers.index(17) # look for an element index
4
>>> numbers.remove(38) # remove an element
>>> numbers
[12, 72, 25, 17, 10]
>>> del numbers[1:3]
>>> numbers
[12, 17, 10]
```

Copy of a list

A new reference to the same list not a copy

```
>>> fable = ['fold', 'but','do’, ‘not','break','point']
>>> sentence = fable
>>> sentence
['fold', 'but', 'do’, ‘not', 'break', 'point']
>>> fable[4] =‘breaking'
>>> fable
['fold', 'but', 'do’, ‘not', 'breaking', 'point']>>>
sentence
['fold', 'but', 'do’, ‘not', 'breaking', 'point']
```

Copy a list:

```
>>> copy = []
>>> for val in fable:
        copy.append(val)
>>> copy
['fold', 'but', 'do’, ‘not', 'breaking', 'point']
>>> copy[4] = ‘break' # does not affect fable
```

Sub-theme: Tuples

- Python offers a data type called tuple, similar to lists but like chains are not changeable.
- A tuple is a collection of elements separated by commas.

```
>>> tup = 'a', 'b', 'c', 'd', 'e'
>>> print(tup)
('a', 'b', 'c', 'd', 'e')
>>> tup[0]
'a'
>>> tup2 = (1, 2, 'c') # syntax alternative more readable
```

- Tuples are preferable to lists whenever we want to be sure that some transmitted data are not modified by error.
- Tuples take less memory space, and could be processed quicker by the interpretor.

Tuples Operations

```
>>> print(tup[2:4])
('c', 'd')
>>> tup[1:3] = ('x', 'y') # error !
'tuple' object does not support item assignment
>>> tup = ('Andre',) + tup[1:]
>>> print(tup)
('Andre', 'b', 'c', 'd', 'e‘)

>>> tu1, tu2 = ("a","b"), ("c","d","e")
>>> tu3 = tu1*4 + tu2    # new tuple
>>> tu3
('a', 'b', 'a', 'b', 'a', 'b', 'a', 'b', 'c', 'd', 'e')

>>> for e in tu3:
        print(e, end=":")
a:b:a:b:a:b:a:b:c:d:e:

>>> del tu3[2]  # error!
'tuple' object doesn't support item deletion
```

Sub-theme: Dictionaries

- The composite data types chains, lists and tuples are sequences of ordered element suites. We can access any element through an index, as long as we know its location. Otherwise you need to visit the sequence element by element.
- The ***dictionaries*** is another composit. They are similar to lists (changeale) but are not sequences.
- The elements that we would record are not in an immutable order. We can access to any one of them using a specific index called ***key*** (could be alphabetical, numrrical, or even a composit type under some conditions).

Creation of a dictionary

```
>>> dico = {}
>>> dico['computer'] = 'ordinateur'
>>> dico['mouse'] ='souris'
>>> dico['keyboard'] ='clavier'
>>> print(dico)
{'computer': 'ordinateur', 'keyboard': 'clavier',
'mouse': 'souris'}
>>> d1 = {1:10, 2:20}
>>> d2 = {'a': 1, 'b': 2, 'c': 3}
```

- Each element is made of a pair of objects : an index (ckey) and a value, separated by a double point.
- Direct and fast access to an element through its key.
- To add un new element, create a new pair key-value. The order is not important.

```
>>> print(dico['mouse'])
souris
```

Creation of a dictionary

- Example: inventory of a fruit batch

```
>>> invent = {apples': 430, 'bananas': 312,
'oranges' : 274, 'peaches' : 137}
>>> print(invent)
{'oranges': 274, ‘apples': 430, 'bananas': 312,
‘peaches': 137}
```

- The boss decides to liquidate all the apples and stop selling them:

```
>>> del invent[‘apples']
>>> print(invent)
{'oranges': 274, 'bananas': 312, 'peaches': 137}
>>> print(len(invent))
3
```

Test of membership in dictionaries

```
>>> if “apples" in invent:
        print(« we have apples")
    else:
        print(« Sorry, no apples.")

Sorry, no apples.

>>> invent['bananas']  # how many bananas?
312                    # fast and direct access
```

Dictionaries are objects

We can use methods such as keys(), values(), items()

```
>>> print(dico.keys())
dict_keys(['computer', 'mouse', 'keyboard'])
>>> for k in dico.keys():
     print(« key :", k, " --- value :", dico[k])

key : computer --- value : ordinateur
key : mouse --- value : souris
key : keyboard --- value : clavier
>>> list(dico.keys())
['computer', 'mouse', 'keyboard']
>>> tuple(dico.keys())
('computer', 'mouse', 'keyboard')
>>> print(invent.values())
dict_values([274, 312, 137])
```

Copying a dictionary

```
>>> stock = invent  # two references to the same
>>> stock           # dictionnary
{'oranges': 274, 'bananas': 312, 'peaches': 137}
>>> del invent['bananas']  # both are modified
>>> stock
{'oranges':  274, 'peaches': 137}

>>> store = stock.copy()  # a copy
>>> store['prunes'] = 561
>>> store
{'oranges': 274, 'prunes': 561, 'peaches': 137}
>>> stock
{'oranges': 274, 'peaches': 137}
>>> invent
{'oranges': 274, 'peaches': 137}
```

going through a dictionary

With for loops:

```
>>> for key in invent:
      print(key, invent[key])

peaches 137
oranges 274

>>> for key, value in invent.items():
       print(key, value)

peaches 137
oranges 274
```

Question:

What line will cause an error?

```
def f(x):
    index = 0
    while index < len(x):
      x[index] = x[index] + 1
      index += 1

t = (10,20,30,40)
f(t)
print(t)
```

Possibles responses (choose one):

a) Line 3 while index < len(x):
b) Line 4 x[index] = x[index] + 1
c) Line 5 index += 1
d) None

Question – *Solution*:

What line will cause an error?

```
def f(x):
    index = 0
    while index < len(x):
      x[index] = x[index] + 1
      index += 1

t = (10,20,30,40)
f(t)
print(t)
```

Correcte response: b)

Explaination: a turple can not be modified

Possibles responses (choose one):

a) Line 3 while index < len(x):
b) Line 4 x[index] = x[index] + 1
c) Line 5 index += 1
d) none

Sets in Python

- A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().
- It can have any number of items and they may be of different types (integer, float, tuple, string etc.).
- A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().
- It can have any number of items and they may be of different types (integer, float, tuple, string etc.).
- But a set cannot have a mutable element, like list or dictionary, as its element.

Sets in Python

```
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set) -->  {1.0, 'Hello', (1, 2, 3)}


# set do not have duplicates
my_set = {1,2,3,4,3,2}
print(my_set) --> {1, 2, 3, 4}


# we can make set from a list
my_set = set([1,2,3,2])
print(my_set) --> {1, 2, 3}
```

Sets in Python

- Creating an empty set is a bit tricky.
- Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the set() function without any argument.

```
# initialize a with {}
a = {}

# check data type of a
print(type(a)) → 'dict'
```

Sets in Python

```
# initialize a with set()
a = set()

# check data type of a
print(type(a))
```
→ ‘set’

- How to change a set in Python?
- Sets are mutable. But since they are unordered, indexing have no meaning.

Sets in Python

- We cannot access or change an element of set using indexing or slicing. Set does not support it.

- We can add single element using the add() method and multiple elements using the update() method. The update() method can take tuples, lists, strings or other sets as its argument.

- In all cases, duplicates are avoided.

Sets in Python

```
# initialize my_set
my_set = {1,3}
print(my_set)

# if you uncomment line my_set[0] below,
# you will get an error
# TypeError: 'set' object does not support indexing
#my_set[0]

# add an element
my_set.add(2)
print(my_set) ==>  {1, 2, 3}
```

Sets in Python

```
# add multiple elements
my_set.update([2,3,4])
print(my_set)  -->  {1, 2, 3, 4}


# add list and set
my_set.update([4,5], {1,6,8})
print(my_set) -->  {1, 2, 3, 4, 5, 6, 8}
```

Sets in Python

- How to remove elements from a set?

- A particular item can be removed from set using methods, discard() and remove().

- The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

Sets in Python

```
# discard an element
# not present in my_set
my_set.discard(2)
print(my_set) -->  my_set.discard(2)


# remove an element
# not present in my_set
# If you uncomment line 27 below,
# you will get an error.
# Output: KeyError: 2

#my_set.remove(2) line 27
```

Sets in Python

- Similarly, we can remove and return an item using the pop() method.

- Set being unordered, there is no way of determining which item will be popped. It is completely arbitrary.

- We can also remove all items from a set using clear().

initialize my_set

- # Output: set of unique elements
- my_set = set("HelloWorld")
- print(my_set) --> {'o', 'e', 'r', 'W', 'd', 'l', 'H'}

Sets in Python

```
# pop an element
# Output: random element
print(my_set.pop()) -->  {'e', 'r', 'W', 'd', 'l', 'H'}

# pop another element
# Output: random element
my_set.pop()
print(my_set) -->  {'r', 'W', 'd', 'l', 'H'}

# clear my_set
my_set.clear()
print(my_set) -->  set()
```

Sets in Python

- We looked at more data types.
- Tuples are like lists but cannot be modified.
- Dictionaries have keys and values, they offer a direct access to keys.